Q.2 a. Explain the database management system and discuss the advantages of database management system and when not to use a DBMS. (8)

Answer: Advantages of Database Management System:

1. Controlling Data Redundancy:
In non-database systems (traditional computer file processing), each application program has its own files. In this case, the duplicated copies of the same data are created at many places. In DBMS, all the data of an organization is integrated into a single database. The data is recorded at only one place in the database and it is not duplicated. For example, the dean's faculty file and the faculty payroll file contain several items that are identical. When they are converted into database, the data is integrated into a single database so that multiple copies of the same data are reduced to-single copy.
In DBMS, the data redundancy can be controlled or reduced but is not removed completely. Sometimes, it is necessary to create duplicate copies of the same data items in order to relate tables with each other.
By controlling the data redundancy, you can save storage space. Similarly, it is useful or retrieving data from database using queries.

2. Data Consistency:
By controlling the data redundancy, the data consistency is obtained. If a data item appears only once, any update to its value has to be performed only once and the updated value (new value of item) is immediately available to all users.
If the DBMS has reduced redundancy to a minimum level, the database system enforces consistency. It means that when a data item appears more than once in the database and is updated, the DBMS automatically updates each occurrence of a data item in the database.

3. Data Sharing:
In DBMS, data can be shared by authorized users of the organization. The DBA manages the data and gives rights to users to access the data. Many users can be authorized to access the same set of information simultaneously. The remote users can also share same data. Similarly, the data of same database can be shared between different application programs.

4. Data Integration:
In DBMS, data in database is stored in tables. A single database contains multiple tables and relationships can be created between tables (or associated data entities). This makes easy to retrieve and update data.

5. Integrity Constraints:
Integrity constraints or consistency rules can be applied to database so that the correct data can be entered into database. The constraints may be applied to data item within a single record or they may be applied to relationships between records.

6. Data Security:
Data security is the protection of the database from unauthorized users. Only the authorized persons are allowed to access the database. Some of the users may be allowed to access only a part of database i.e., the data that is related to them or related to their department. Mostly, the DBA or head of a department can access all the data in the database. Some users may be permitted only to retrieve data, whereas others are allowed to retrieve as well as to update data. The database access is controlled by the DBA. He creates the accounts of users and gives rights

to access the database. Typically, users or group of users are given usernames protected by passwords.

Disadvantages of Database Management System (DBMS):

1. Cost of Hardware & Software:

A processor with high speed of data processing and memory of large size is required to run the DBMS software. It means that you have to upgrade the hardware used for file-based system. Similarly, DBMS software is also Very costly.

2. Cost of Data Conversion:

When a computer file-based system is replaced with a database system, the data stored into data file must be converted to database files. It is difficult and time consuming method to convert data of data files into database. You have to hire DBA (or database designer) and system designer along with application programmers; Alternatively, you have to take the services of some software houses. So a lot of money has to be paid for developing database and related software.

3. Cost of Staff Training:

Most DBMSs are often complex systems so the training for users to use the DBMS is required. Training is required at all levels, including programming, application development, and database administration. The organization has to pay a lot of amount on the training of staff to run the DBMS.

4. Appointing Technical Staff:

The trained technical persons such as database administrator and application programmers etc are required to handle the DBMS. You have to pay handsome salaries to these persons. Therefore, the system cost increases.

5. Database Failures:

In most of the organizations, all data is integrated into a single database. If database is corrupted due to power failure or it is corrupted on the storage media, then our valuable data may be lost or whole system stops.

b. Discuss the centralized and client/server architectures for DBMS. (8)

Answer: In **centralized database systems**, the database system, application programs, and user-interface all are executed on a single system and dummy terminals are connected to it. The processing power of single system is utilized and dummy terminals are used only to display the information. As the personal computers became faster, more powerful, and cheaper, the database system started to exploit the available processing power of the system at the user's side, which led to the development of client/server architecture. In **client/server architecture**, the processing power of the computer system at the user's end is utilized by processing the user-interface on that system.

A **client** is a computer system that sends request to the server connected to the network, and a **server** is a computer system that receives the request, processes it, and returns the requested information back to the client. Client and server are usually present at different sites. The end users (remote database users) work on client computer system and database system runs on the server. Servers can be of several types, for example, file servers, printer servers, web servers, database servers, etc. The client machines have user interfaces that help users to utilize the servers. It also provides users the local processing power to run local applications on the client side.

There are two approaches to implement client/server architecture. In the first approach, the user interface and application programs are placed on the client side and the database system on the server side. This architecture is called **two-tier architecture**. The application programs that reside at the client side invoke the DBMS at the server side. The application program interface standards like Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC) are used for interaction between client and server.

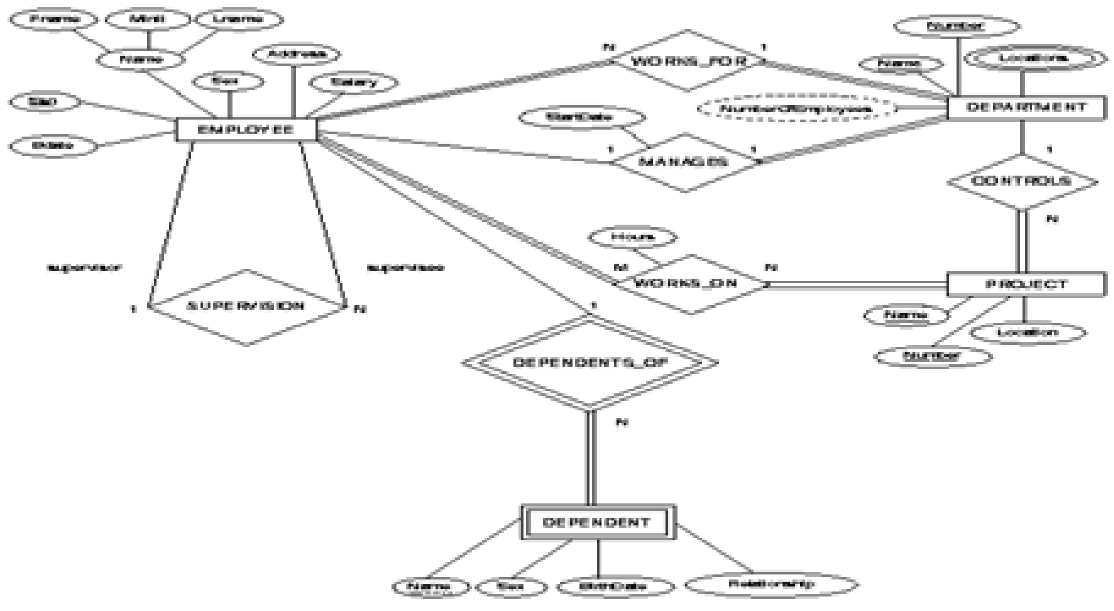
The second approach, that is, **three-tier architecture** is primarily used for web-based applications. It adds intermediate layer known as **application server** (or **web server**) between the client and the database server. The client communicates with the application server, which in turn communicates with the database server. The application server stores the business rules (procedures and constraints) used for accessing data from database server. It checks the client's credentials before forwarding a request to database server. Hence, it improves database security.

When a client requests for information, the application server accepts the request, processes it, and sends corresponding database commands to database server. The database server sends the result back to application server which is converted into GUI format and presented to the client.

Q.3 a. Draw an Entity-Relation (E-R) diagram for a company database. Assume desired entities, attributes and relations for the company database by yourself and mention all at the starting. (6)

Answer:

Figure 3.2



© The Benjamin/Cummings Publishing Company, Inc. 1994, Elmasri/Navathe, Fundamentals of Database Systems, Second Edition

b. What is the difference between weak and strong entity set? Explain with example. (4)

Answer: An entity set that does not possess sufficient attributes to form a primary key is called a weak entity set. One that does have a primary key is called a strong entity set. For example,

The entity set transaction has attributes transaction-number, date and amount.
Different transactions on different accounts could share the same number.
These are not sufficient to form a primary key (uniquely identify a transaction).
Thus transaction is a weak entity set.
For a weak entity set to be meaningful, it must be part of a one-to-many relationship set. This relationship set should have no descriptive attributes.

The idea of strong and weak entity sets is related to the existence dependencies seen earlier.

Member of a strong entity set is a dominant entity. Member of a weak entity set is a subordinate entity. A weak entity set does not have a primary key, but we need a means of distinguishing among the entities. The discriminator of a weak entity set is a set of attributes that allows this distinction to be made.
The primary key of a weak entity set is formed by taking the primary key of the strong entity set on which its existence depends (see Mapping Constraints) plus its discriminator.

c. Define and explain the different types of relationships exist in the DBMS. (6)

Answer: Relationship and types of Relationships:
A relationship describes an association among entities. For example, a relationship exists between customers and agents that can be described as follows: an agent can serve many customers, and each customer may be served by one agent. Data models use three types of relationships: one-to-many, many-to-many, and one-to-one. Database designers usually use the shorthand notations 1:M or 1..*, M:N or *..*, and 1:1 or 1..1, respectively. (Although the M:N notation is a standard label for the many-to-many relationship, the label M:M may also be used.) The following examples illustrate the distinctions among the three.

One-to-many (1:M or 1..*) relationship:
A painter paints many different paintings, but each one of them is painted by only one painter. Thus, the painter (the "one") is related to the paintings (the "many"). Therefore, database designers label the relationship "PAINTER paints PAINTING" as 1:M. (Note that entity names are often capitalized as a convention, so they are easily identified.) Similarly, a customer (the "one") may generate many invoices, but each invoice (the "many") is generated by only a single customer. The "CUSTOMER generates INVOICE" relationship would also be labeled 1:M.

Many-to-many (M:N or *..*) relationship:
An employee may learn many job skills, and each job skill may be learned by many employees. Database designers label the relationship "EMPLOYEE learns SKILL" as M:N. Similarly, a student can take many classes and each class can be taken by many students, thus yielding the M:N relationship label for the relationship expressed by "STUDENT takes CLASS."

One-to-one (1:1 or 1..1) relationship:
A retail company's management structure may require that each of its stores be managed by a single employee. In turn, each store manager, who is an employee, manages only a single store. Therefore, the relationship "EMPLOYEE manages STORE" is labeled 1:1.

Q.4 a. Consider the relations: (6)
PROJECT(proj#,proj_name,chief_architect)
EMPLOYEE(emp#,emp_name)
ASSIGNED(proj#,emp#)
Use relational algebra to express the following queries:
(i) Get details of employees working on project COMP33.
(ii) Get details of project on which employee with name 'RAM' is working.
(iii) find project name whose chief architect name is RAJ.

Answer: i) $\pi_{emp\_name}(\sigma_{proj\_name=”COMP33”}$(PROJECT× EMPLOYEE × ASSIGNED)

ii) $\pi_{proj\_name,\ proj\#}(\sigma_{emp\_name=\ “RAM”}$(PROJECT× EMPLOYEE × ASSIGNED)

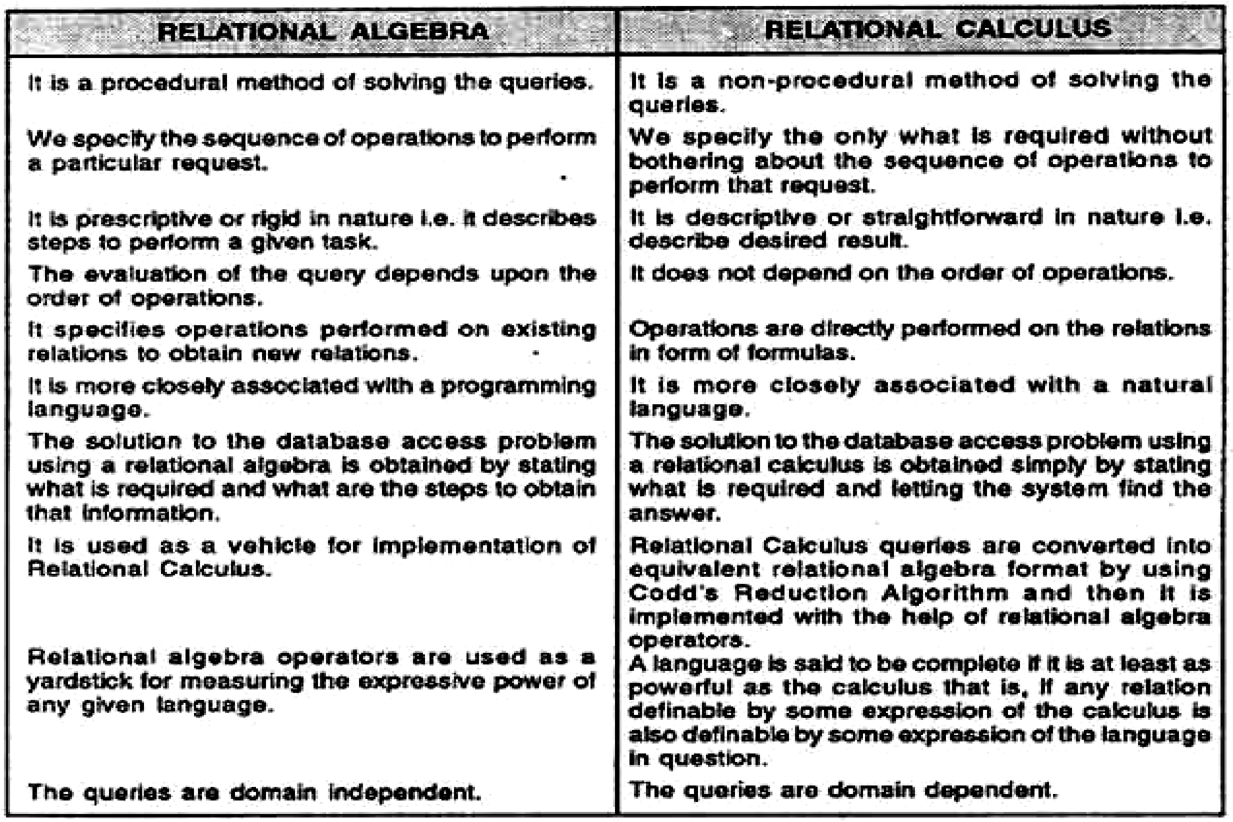
iii) π proj_name($\sigma_{chief\_architect=’”RAJ”}$(PROJECT)

b. Differentiate between inner join and outer join. (4)

Answer: An **inner join** will only select records where the joined keys are in **both** specified tables.
A **left outer join** will select all records from the first table, and any records in the second table that match the joined keys.
A **right outer join** will select all records from the second table, and any records in the first table that match the joined keys.

c. What is the basic differences between relational algebra and relational calculus? (6)

Answer:



RELATIONAL ALGEBRA	RELATIONAL CALCULUS
It is a procedural method of solving the queries.	It is a non-procedural method of solving the queries.
We specify the sequence of operations to perform a particular request.	We specify the only what is required without bothering about the sequence of operations to perform that request.
It is prescriptive or rigid in nature i.e. it describes steps to perform a given task.	It is descriptive or straightforward in nature i.e. describe desired result.
The evaluation of the query depends upon the order of operations.	It does not depend on the order of operations.
It specifies operations performed on existing relations to obtain new relations.	Operations are directly performed on the relations in form of formulas.
It is more closely associated with a programming language.	It is more closely associated with a natural language.
The solution to the database access problem using a relational algebra is obtained by stating what is required and what are the steps to obtain that information.	The solution to the database access problem using a relational calculus is obtained simply by stating what is required and letting the system find the answer.
It is used as a vehicle for implementation of Relational Calculus.	Relational Calculus queries are converted into equivalent relational algebra format by using Codd's Reduction Algorithm and then it is implemented with the help of relational algebra operators.
Relational algebra operators are used as a yardstick for measuring the expressive power of any given language.	A language is said to be complete if it is at least as powerful as the calculus that is, if any relation definable by some expression of the calculus is also definable by some expression of the language in question.
The queries are domain independent.	The queries are domain dependent.

Q.5 a. Discuss the various normal forms upto BCNF for normalizing a relation with suitable examples. (8)

Answer:

First Normal Form:

First normal form (1NF or Minimal Form) is a normal form used in database normalization. A relational database table that adheres to 1NF is one that meets a certain minimum set of criteria. These criteria are basically concerned with ensuring that the table is a faithful representation of a relation and that it is free of repeating groups. The concept of a "repeating group" is, however, understood in different ways by different theorists. As a consequence, there is no universal agreement as to which features would disqualify a table from being in 1NF.

Examples of tables (or views) that would not meet this definition of 1NF are:

• A table that lacks a unique key. Such a table would be able to accommodate duplicate rows, in violation of condition

• A view whose definition mandates that results be returned in a particular order, so that the row-ordering is an intrinsic and meaningful aspect of the view. This violates condition 1. The tuples in true relations are not ordered with respect to each other.

• A table with at least one null able attributes. A null able attribute would be in violation of condition 4, which requires every field to contain exactly one value from its column's domain. It should be noted, however, that this aspect of condition 4 is controversial. It marks an important departure from Codd's later vision of the relational model, which made explicit provision for nulls.

Strictly speaking, NF1 addresses two issues:

1. A row of data cannot contain repeating groups of similar data (atomicity); and

2. Each row of data must have a unique identifier.

Second Normal Form:

Second normal form (2NF) is a normal form used in database normalization. 2NF was originally defined by E.F. Codd[1] in 1971. A table that is in first normal form (1NF) must meet additional criteria if it is to qualify for second normal form. Specifically: a 1NF table is in 2NF if and only if, given any candidate key and any attribute that is not a constituent of a candidate key, the non-key attribute depends upon the whole of the candidate key rather than just a part of it.

In slightly more formal terms: a 1NF table is in 2NF if and only if none of its non-prime attributes are functionally dependent on a part (proper subset) of a candidate key. (A non-prime attribute is one that does not belong to any candidate key.) Note that when a 1NF table has no composite candidate keys (candidate keys consisting of more than one attribute), the table is automatically in 2NF.

Third Normal Form:

The third normal form (3NF) is a normal form used in database normalization. 3NF was originally defined by E.F. Codd[1] in 1971. Codd's definition states that a table is in 3NF if and only if both of the following conditions hold:

The relation R (table) is in second normal form (2NF)

Every non-prime attribute of R is non-transitively dependent (i.e. directly dependent) on every key of R. A non-prime attribute of R is an attribute that does not belong to any candidate key of R.[2] A transitive dependency is a functional dependency in which X → Z (X determines Z) indirectly, by virtue of X → Y and Y → Z (where it is not the case that Y → X).[3]

A 3NF definition that is equivalent to Codd's, but expressed differently, was given by Carlo Zaniolo in 1982. This definition states that a table is in 3NF if and only if, for each of its functional dependencies X → A, at least one of the following conditions holds:

X contains A (that is, X → A is trivial functional dependency), or

X is a superkey, or

A is a prime attribute (i.e., A is contained within a candidate key)

Zaniolo's definition gives a clear sense of the difference between 3NF and the more stringent Boyce-Codd normal form (BCNF). BCNF simply eliminates the third alternative ("A is a prime attribute").

Normalization beyond 3NF, Most 3NF tables are free of update, insertion, and deletion anomalies. Certain types of 3NF tables, rarely met with in practice, are affected by such anomalies; these are tables which either fall short of Boyce-Codd normal form (BCNF) or, if they meet BCNF, fall short of the higher normal forms 4NF or 5NF.

b. Define the followings: **(4)**
i) Multivalued dependencies
ii) Join Dependencies

Answer:
i) **Multivalued dependencies**
Multivalued dependencies occur when the presence of one or more rows in a table implies the presence of one or more other rows in that same table.

Examples:
For example, imagine a car company that manufactures many models of car, but always makes both red and blue colors of each model. If you have a table that contains the model name, color and year of each car the company manufactures, there is a multivalued dependency in that table. If there is a row for a certain model name and year in blue, there must also be a similar row corresponding to the red version of that same car.

ii) **Join Dependency**
A *join dependency* (*JD*) can be said to exist if the join of R_1 and R_2 over C is equal to relation R. Where, R_1 and R_2 are the decompositions $R_1(A, B, C)$, and R_2 (C, D) of a given relations R (A, B, C, D). Alternatively, R_1 and R_2 is a lossless decomposition of R. In other words, $*(A, B, C, D), (C, D)$ will be a join dependency of R if the join of the join's attributes is equal to relation R. Here, $*(R_1, R_2, R_3, ...)$ indicates that relations R_1, R_2, R_3 and so on are a join dependency (JD) of R.

c. Given R(A,B,C,D,E) with the set of FDs, (4)
F{AB→CD, ABC → E, C → A}
(i) Find any two candidate keys of R
(ii) What is the normal form of R? Justify your answer.

Answer:
(i) To find two candidate keys of R, we have to find the closure of the set of attributes under consideration and if all the attributes of R are in the closure then that set is a candidate key. Now from the set of FD's we can make out that B is not occurring on the RHS of any FD, therefore, it must be a part of the candidate keys being considered otherwise it will not be in the closure of any attribute set. So let us consider the following sets AB and BC.
Now (AB)+= ABCDE, CD are included in closure because of the FD AB → CD, and E is included in closure because of the FD ABC → E.
Now (BC)+= BCAED, A is included in closure because of the FD C → A, and then E is included in closure because of the FD ABC → E and lastly D is included in closure because of the FD AB → CD.

(ii) The prime attributes are A, B and C and non-prime attributes are D and E.
A relation scheme is in 2NF, if all the non-prime attributes are fully functionally dependent on the relation key(s). From the set of FDs we can see that the non-prime attributes (D,E) are fully functionally dependent on the prime attributes, therefore, the relation is in 2NF.
A relation scheme is in 3NF, if for all the non-trivial FDs in F+ of the form X → A, either X is a superkey or A is prime. From the set of FDs we see that for all the FDs, this is satisfied, therefore, the relation is in 3NF.
A relation scheme is in BCNF, if for all the non-trivial FDs in F+ of the form X→ A, X is a superkey. From the set of FDs we can see that for the FD C → A, this is not satisfied as LHS is not a superkey, therefore, the relation is not in BCNF. Hence, the given relation scheme is in 3NF.

Q.6 a. Discuss various indexing attributes based on indexing. (6)

Answer: Indexing is defined based on its indexing attributes. Indexing can be one of the following types:
Primary Index: If index is built on ordering 'key-field' of file it is called Primary Index. Generally it is the primary key of the relation.
Secondary Index: If index is built on non-ordering field of file it is called Secondary Index.
Clustering Index: If index is built on ordering non-key field of file it is called Clustering Index.

Ordering field is the field on which the records of file are ordered. It can be different from primary or candidate key of a file.

b. What do you understand by RAID? Explain RAID Level5. (5)

Answer: RAID stands for redundant array of independent disks. The name indicates that the disk drives are independent, and are multiple in number. How the data is distributed between these drives depends on the RAID level used.
The main advantage of RAID, is the fact that, to the operating system the array of disks can be presented as a single disk.

RAID is fault tolerant because in most of the RAID level's data is redundant in multiple disks, so even if one disk fails, or even two sometimes, the data will be safe and the operating system will not be even aware of the failure. DATA loss is prevented due to the fact that data can be recovered from the disk that is not failed.
RAID level 5 uses striping, so data is spread across number of disks used in the array, and also provides redundancy with the help of parity.
RAID 5 is a best cost effective solution for both performance and redundancy. Striped method of storing data always improves performance, and parity used in this level of raid is distributed parity.
Minimum number of disks required for RAID 5 is 3, and maximum can go upto 32(depending on the RAID controller used.)
One important fact to note is that, reading rate in RAID 5 is much better than writing. This is because reading can be done, by a combined rate of all disks used.
As a reference you can have a look at the distributed parity diagram shown in the **Parity in Raid section** of this article.

c. Define two-phase locking protocol. (5)

Answer: In databases and transaction processing, **two-phase locking** (**2PL**) is a concurrency control method that guarantees serializability.[1][2] It is also the name of the resulting set of database transaction schedules (histories). The protocol utilizes locks, applied by a transaction to data, which may block (interpreted as signals to stop) other transactions from accessing the same data during the transaction's life.

By the 2PL protocol locks are applied and removed in two phases:
1. Expanding phase: locks are acquired and no locks are released.
2. Shrinking phase: locks are released and no locks are acquired.

Q.7 a. Describe the nested-loop join and block-nested loop join. Compare them.(8)

Answer: [1] Nested Loop Join
Number of tuples in R * Number of Blocks in S + Number of Blocks in R. Here choosing relation with small number of tuples as outer relation R
400 * 80 + 20
therefore ans :[c] 32020

[2]Block Nested Loop Join
Number of Blocks in R * Number of Blocks in S + Number of Blocks in R Here choosing relation with small number of Blocks as outer relation R
20 * 80 + 20 = 1620
difference : 32020 – 1620
therefore ans:[b] 30400

b. Draw a state diagram and discuss the typical states that a transaction goes through during execution. (3)

Answer:

A transaction is an atomic unit of work that is either completed in its entirety or not done at all. For recovery purposes, the system needs to keep track of when the transaction starts, terminates, and commits or aborts (see Section 17.2.3). Therefore, the recovery manager keeps track of the following operations:

- BEGIN_TRANSACTION. This marks the beginning of transaction execution.
- READ OR WRITE. These specify read or write operations on the database items that are executed as part of a transaction.
- END_TRANSACTION. This specifies that READ and WRITE transaction operations have ended and marks the end of transaction execution. However, at this point it may be necessary to check whether the changes introduced by the transaction can be permanently applied to the database (committed) or whether the transaction has to be aborted because it violates serializability (see Section 17.5) or for some other reason.
- COMMIT_TRANSACTION. This signals a *successful end* of the transaction so that any changes (updates) executed by the transaction can be safely **committed** to the database and will not be undone.
- ROLLBACK (or ABORT). This signals that the transaction has *ended unsuccessfully*, so that any changes or effects that the transaction may have applied to the database must be *undone*.

Figure 17.4 shows a state transition diagram that describes how a transaction moves through its execution states. A transaction goes into an **active state** immediately after it starts execution, where it can issue READ and WRITE operations. When the transaction ends, it moves to the **partially committed state**. At this point, some recovery protocols need to ensure that a system failure will not result in an inability to record the changes of the transaction permanently (usually by recording changes in the system log, discussed in the next section).[5] Once this check is successful, the transaction is said to have reached its commit point and enters the **committed state**. Commit points are discussed in more detail in Section 17.2.3. Once a transaction is committed, it has concluded its execution successfully and all its changes must be recorded permanently in the database.

However, a transaction can go to the **failed state** if one of the checks fails or if the transaction is aborted during its active state. The transaction may then have to be rolled back to undo the effect of its WRITE operations on the database. The **terminated state** corresponds to the transaction leaving the system. The transaction information that is maintained in system tables while the transaction has been

5. Optimistic concurrency control (see Section 18.4) also requires that certain checks are made at this point to ensure that the transaction did not interfere with other executing transactions.

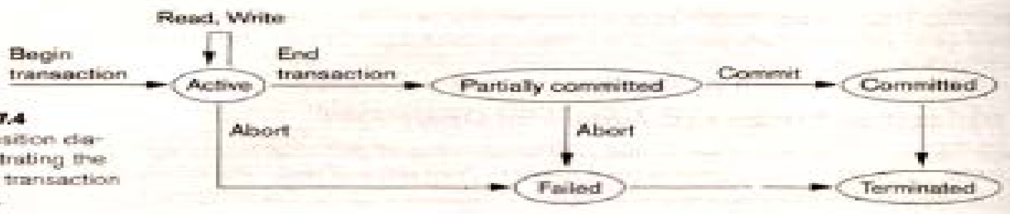


Figure 17.4
State transition diagram illustrating the states for transaction execution.

running is removed when the transaction terminates. Failed or aborted transactions may be *restarted* later—either automatically or after being resubmitted by the user—as brand new transactions.

c. Explain the ACID properties of a transaction. (5)

Answer: Database ACID (Atomicity, Consistency, Isolation, Durability)

Properties

There are a set of properties that guarantee that database transactions are processed reliably, referred to as ACID (Atomicity, Consistency, Isolation, Durability).

Atomicity

Atomicity refers to the ability of the database to guarantee that either all of the tasks of a transaction are performed or none of them are. Database modifications must follow an all or nothing rule. Each transaction is said to be atomic if when one part of the transaction fails, the entire transaction fails.

Consistency

The consistency property ensures that the database remains in a consistent state before the start of the transaction and after the transaction is over (whether successful or not). For example, in a storefront there is an inconsistent view of what is truly available for purchase if inventory is allowed to fall below 0, making it impossible to provide more than an intent to complete a transaction at checkout time. An example in a double-entry accounting system illustrates the concept of a true transaction. Every debit requires an associated credit. Both of these happen or neither happen.

A distributed data system is either strongly consistent or has some form of weak consistency. Once again, using the storefront example, a database needs to provide consistency and isolation, so that when one customer is reducing an item in stock and in parallel is increasing the basket by one, this is isolated from another customer who will have to wait while the data store catches up. At the other end of the spectrum is BASE (Basically Available Soft-state Eventual consistency).

Weak consistency is sometimes referred to as eventual consistency, the database eventually reaches a consistent state. Weak consistency systems are usually ones where data is replicated; the latest version is sitting somewhere in the cluster, older versions are still out there. Eventually all nodes will see the latest version.

Isolation

Isolation refers to the requirement that other operations cannot access or see the data in an intermediate state during a transaction. This constraint is required to maintain the performance as well as the consistency between transactions in a database. Thus, each transaction is unaware of another transactions executing concurrently in the system.

Durability

Durability refers to the guarantee that once the user has been notified of success, the transaction will persist, and not be undone. This means it will survive system failure, and that the database system has checked the integrity constraints and won't need to abort the transaction. Many databases implement durability by writing all transactions into a transaction log that can be played back to recreate the system state right before a failure. A transaction can only be deemed committed after it is safely in the log.

Durability does not imply a permanent state of the database. Another transaction may overwrite any changes made by the current transaction without hindering durability.

Q.8 (For Current Scheme student i.e. AC61/AT61)

a. Compare wait-die deadlock prevention scheme with wait-wound scheme.(8)

Answer: Wait-die scheme: It is a non-preemptive technique for deadlock prevention. When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp smaller than that of T_j (That is T_i is older than T_j), otherwise T_i is rolled back (dies)

For example:

Suppose that transaction T_{22}, T_{23}, T_{24} have time-stamps 5, 10 and 15 respectively. If T_{22}requests a data item held by T_{23} then T_{22} will wait. If T_{24} requests a data item held by T_{23}, then T_{24} will be rolled back.

Wound-wait scheme: It is a preemptive technique for deadlock prevention. It is a counterpart to the wait-die scheme. When Transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j, otherwise T_j is rolled back (T_j is wounded by T_i)

For example:

Suppose that Transactions T_{22}, $T_{23,}$ T_{24} have time-stamps 5, 10 and 15 respectively . If T_{22}requests a data item held by T_{23}, then data item will be preempted from T_{23} and T_{23} will be rolled back. If T_{24} requests a data item held by T_{23}, then T_{24} will wait.

b. Explain the differences between a file-oriented system and a database oriented system. (8)

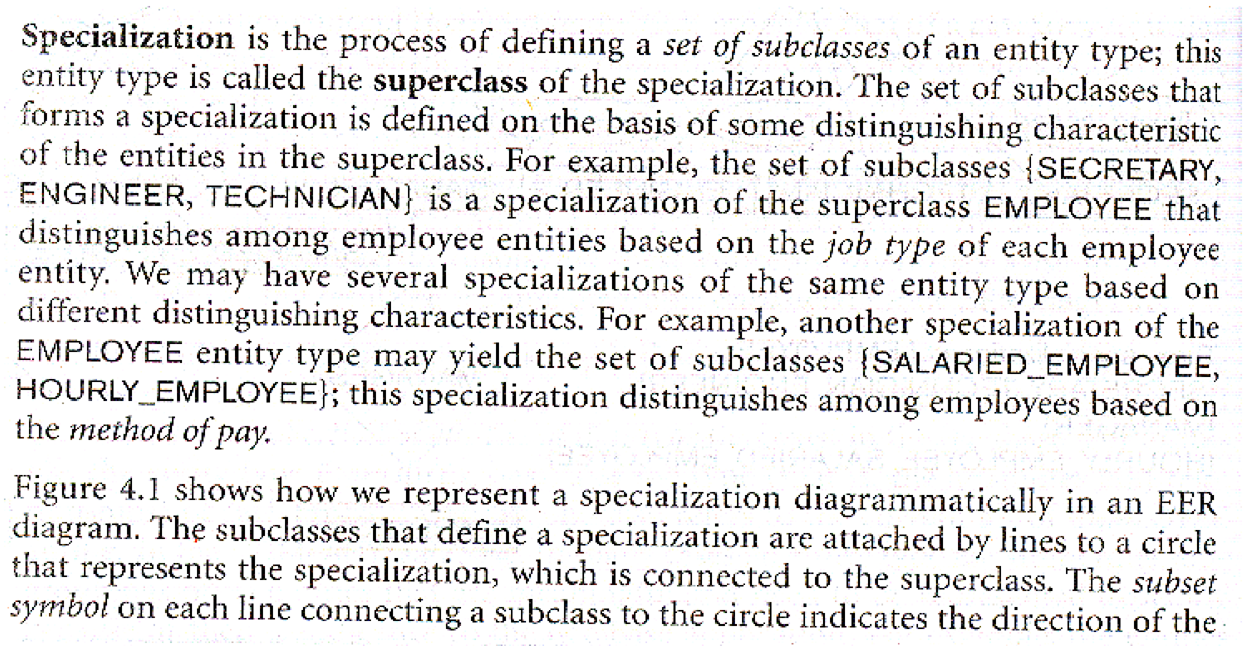
Answer: Computer-based data processing systems were initially used for scientific and engineering calculations. With increased complexity of business requirements, gradually they were introduced into the business applications. The manual method of filing systems of an organisation, such as to hold all internal and external correspondence relating to a project or activity, client, task, product, customer or employee, was maintaining different manual folders. These files or folders were labelled and stored in one or more cabinets or almirahs under lock and key for safety and security reasons. As and when required, the concerned person in the organisation used to search for a specific folder or file serially starting from the first entry. Alternatively, files were indexed to help locate the file or folder more quickly. Ideally, the contents of each file folder were logically related. For example, a file folder in a supplier's office might contain customer data; one file folder for each customer. All data in that folder described only that customer's transaction. Similarly, a personnel manager might organise personnel data of employees by category of employment (for example, technical, secretarial, sales, administrative, and so on). Therefore, a file folder leveled 'technical' would contain data pertaining to only those people whose duties were properly classified as technical.

The manual system worked well as data repository as long as the data collection were relatively small and the organisation's managers had few reporting requirements. However, as the organisation grew and as the reporting requirements became more complex, it became difficult in keeping track of data in the manual file system. Also, report generation from a manual file system could be slow and cumbersome. Thus, this manual filing system was replaced with a computer-based filing system. File-oriented systems were an early attempt to computerize the manual filing system that we are familiar with. Because these systems performed normal record-keeping functions, they were called *data processing (DP) systems*. Rather than establish a centralised store for organisation's operational data, a decentralised approach was taken, where each department, with the assistance of DP department staff, stored and controlled its own data.

Q.8 (For New Scheme student i.e. AC112/AT112)

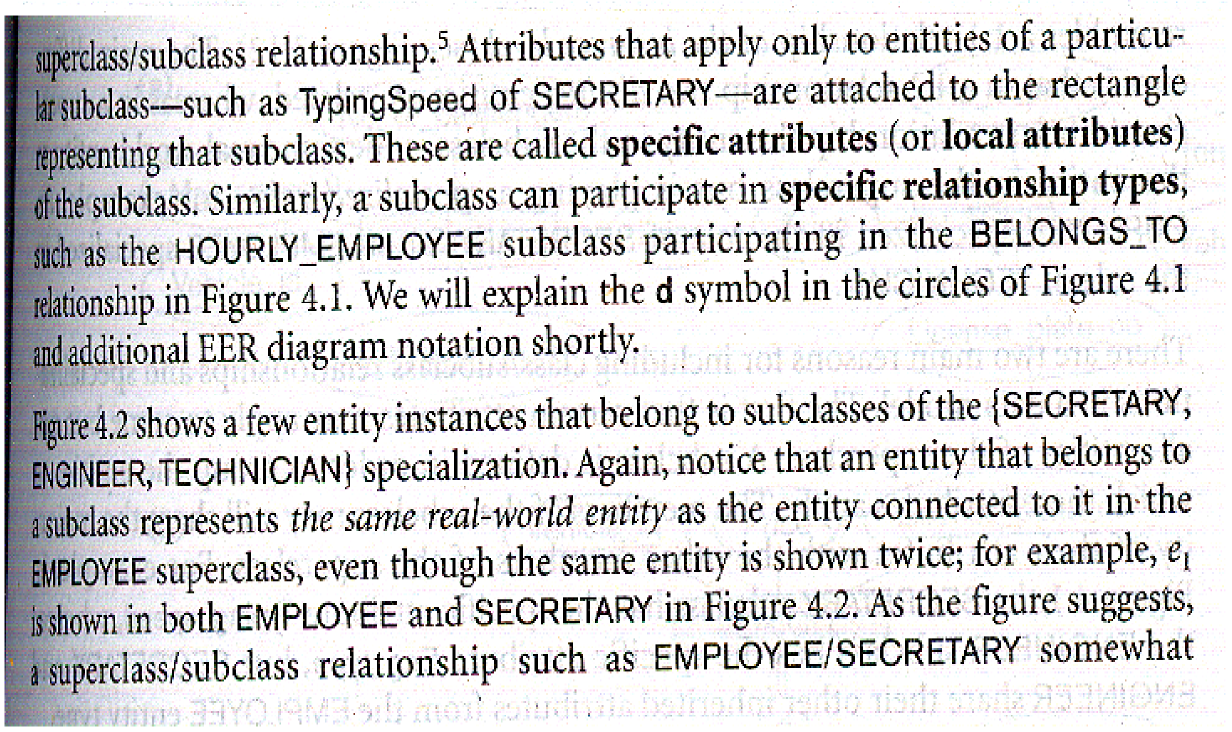
a. Explain generalization and specialization with suitable examples. (6)

Answer:



Specialization is the process of defining a *set of subclasses* of an entity type; this entity type is called the **superclass** of the specialization. The set of subclasses that forms a specialization is defined on the basis of some distinguishing characteristic of the entities in the superclass. For example, the set of subclasses {SECRETARY, ENGINEER, TECHNICIAN} is a specialization of the superclass EMPLOYEE that distinguishes among employee entities based on the *job type* of each employee entity. We may have several specializations of the same entity type based on different distinguishing characteristics. For example, another specialization of the EMPLOYEE entity type may yield the set of subclasses {SALARIED_EMPLOYEE, HOURLY_EMPLOYEE}; this specialization distinguishes among employees based on the *method of pay*.

Figure 4.1 shows how we represent a specialization diagrammatically in an EER diagram. The subclasses that define a specialization are attached by lines to a circle that represents the specialization, which is connected to the superclass. The *subset symbol* on each line connecting a subclass to the circle indicates the direction of the



superclass/subclass relationship.[5] Attributes that apply only to entities of a particular subclass—such as TypingSpeed of SECRETARY—are attached to the rectangle representing that subclass. These are called **specific attributes** (or **local attributes**) of the subclass. Similarly, a subclass can participate in **specific relationship types**, such as the HOURLY_EMPLOYEE subclass participating in the BELONGS_TO relationship in Figure 4.1. We will explain the **d** symbol in the circles of Figure 4.1 and additional EER diagram notation shortly.

Figure 4.2 shows a few entity instances that belong to subclasses of the {SECRETARY, ENGINEER, TECHNICIAN} specialization. Again, notice that an entity that belongs to a subclass represents *the same real-world entity* as the entity connected to it in the EMPLOYEE superclass, even though the same entity is shown twice; for example, e_1 is shown in both EMPLOYEE and SECRETARY in Figure 4.2. As the figure suggests, a superclass/subclass relationship such as EMPLOYEE/SECRETARY somewhat

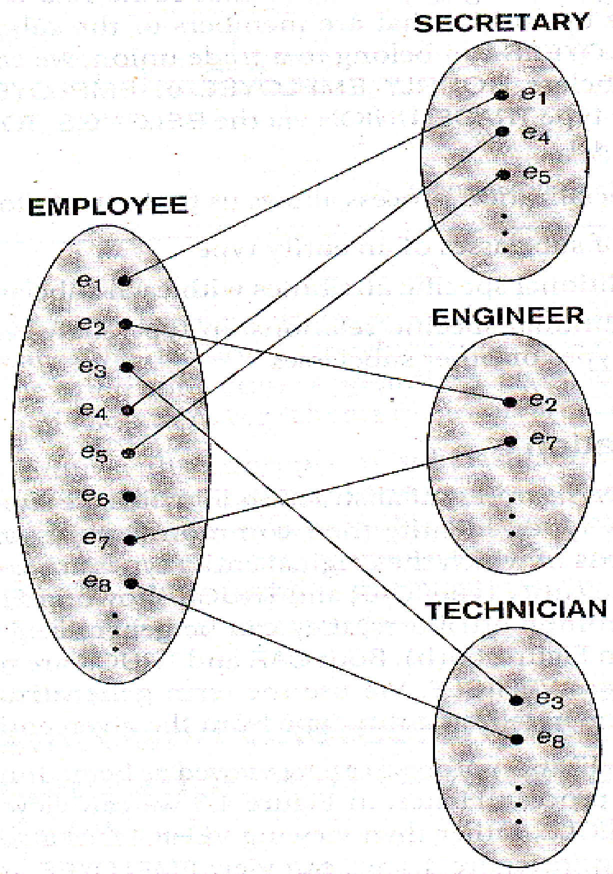


Figure 4.2
Instances of a specialization.

resembles a 1:1 relationship *at the instance level* (see Figure 3.12). The main difference is that in a 1:1 relationship two *distinct entities* are related, whereas in a superclass/subclass relationship the entity in the subclass is the same real-world entity as the entity in the superclass but is playing a *specialized role*—for example, an EMPLOYEE specialized in the role of SECRETARY, or an EMPLOYEE specialized in the role of TECHNICIAN.

There are two main reasons for including class/subclass relationships and specializations in a data model. The first is that certain attributes may apply to some but not all entities of the superclass. A subclass is defined in order to group the entities to which these attributes apply. The members of the subclass may still share the majority of their attributes with the other members of the superclass. For example, in Figure 4.1 the SECRETARY subclass has the specific attribute Typing_speed, whereas the ENGINEER subclass has the specific attribute Eng_type, but SECRETARY and ENGINEER share their other inherited attributes from the EMPLOYEE entity type.

The second reason for using subclasses is that some relationship types may be participated in only by entities that are members of the subclass. For example, if only HOURLY_EMPLOYEES can belong to a trade union, we can represent that fact by creating the subclass HOURLY_EMPLOYEE of EMPLOYEE and relating the subclass to an entity type TRADE_UNION via the BELONGS_TO relationship type, as illustrated in Figure 4.1.

In summary, the specialization process allows us to do the following:

- Define a set of subclasses of an entity type
- Establish additional specific attributes with each subclass
- Establish additional specific relationship types between each subclass and other entity types or other subclasses

4.2.2 Generalization

We can think of a *reverse process* of abstraction in which we suppress the differences among several entity types, identify their common features, and **generalize** them into a single **superclass** of which the original entity types are special **subclasses**. For example, consider the entity types CAR and TRUCK shown in Figure 4.3(a). Because they have several common attributes, they can be generalized into the entity type VEHICLE, as shown in Figure 4.3(b). Both CAR and TRUCK are now subclasses of the **generalized superclass** VEHICLE. We use the term **generalization** to refer to the process of defining a generalized entity type from the given entity types.

Notice that the generalization process can be viewed as being functionally the inverse of the specialization process. Hence, in Figure 4.3 we can view {CAR, TRUCK} as a specialization of VEHICLE, rather than viewing VEHICLE as a generalization of CAR and TRUCK. Similarly, in Figure 4.1 we can view EMPLOYEE as a generalization of SECRETARY, TECHNICIAN, and ENGINEER. A diagrammatic notation to distinguish between generalization and specialization is used in some design methodologies. An arrow pointing to the generalized superclass represents a generalization, whereas

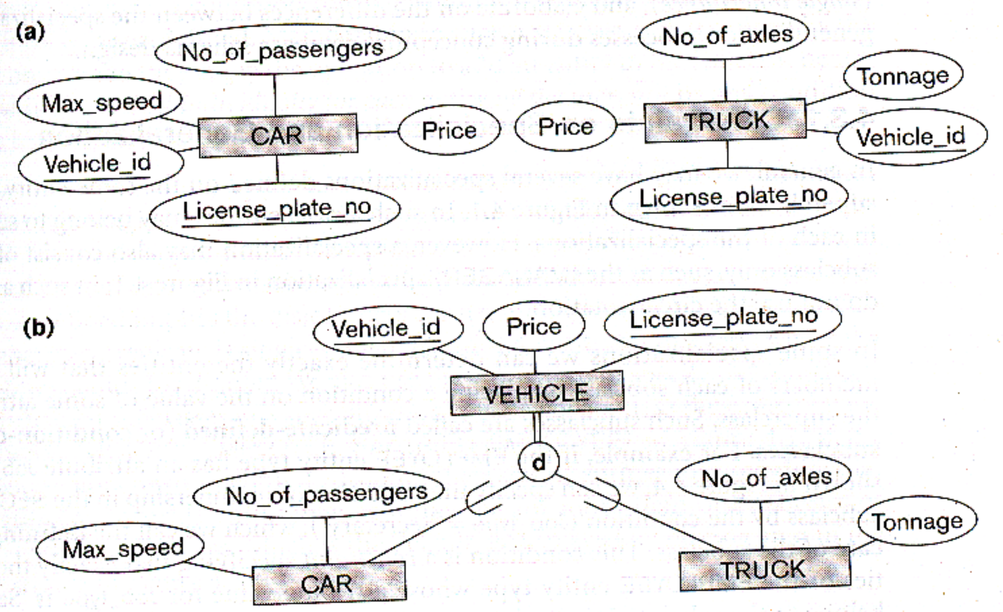


Figure 4.3
Generalization. (a) Two entity types, CAR and TRUCK.
(b) Generalizing CAR and TRUCK into the superclass VEHICLE.

arrows pointing to the specialized subclasses represent a specialization. We will *not* use this notation because the decision as to which process is more appropriate in a particular situation is often subjective. Appendix A gives some of the suggested alternative diagrammatic notations for schema diagrams and class diagrams.

So far we have introduced the concepts of subclasses and superclass/subclass relationships, as well as the specialization and generalization processes. In general, a superclass or subclass represents a collection of entities of the same type and hence also describes an *entity type;* that is why superclasses and subclasses are shown in rectangles in EER diagrams, like entity types. Next, we discuss the properties of specializations and generalizations in more detail.

b. List the advantages of distributed database management system. (5)

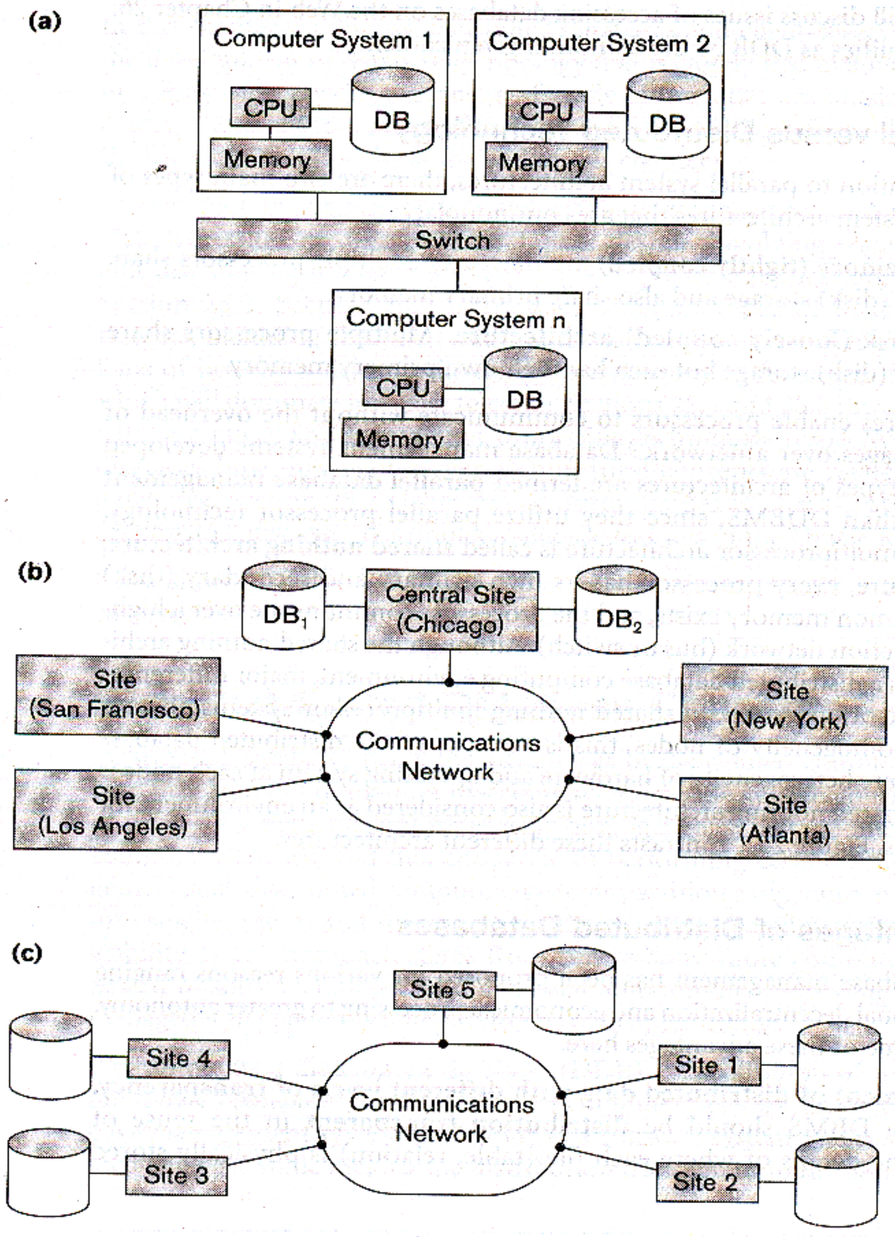
Answer:

25.1.2 Advantages of Distributed Databases

Distributed database management has been proposed for various reasons ranging from organizational decentralization and economical processing to greater autonomy. We highlight some of these advantages here.

1. **Management of distributed data with different levels of transparency.** Ideally, a DBMS should be **distribution transparent** in the sense of hiding the details of where each file (table, relation) is physically stored

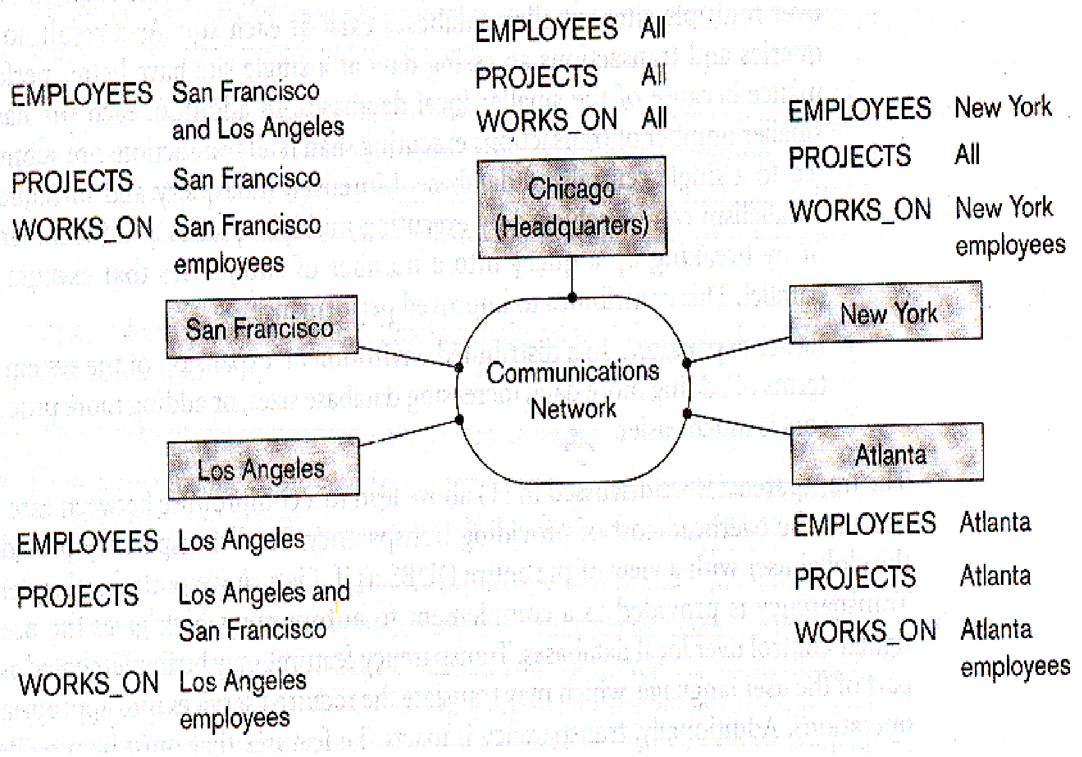
Figure 25.1
Some different database system architectures. (a) Shared nothing architecture. (b) A networked architecture with a centralized database at one of the sites. (c) A truly distributed database architecture.



within the system. Consider the company database in Figure 5.5 that we have been discussing throughout the book. The EMPLOYEE, PROJECT, and WORKS_ON tables may be fragmented horizontally (that is, into sets of rows, as we shall discuss in Section 25.2) and stored with possible replication as shown in Figure 25.2. The following types of transparencies are possible:

- **Distribution or network transparency.** This refers to freedom for the user from the operational details of the network. It may be divided into location transparency and naming transparency. **Location transparency** refers to the fact that the command used to perform a task is independent of the location of data and the location of the system where the command was issued. **Naming transparency** implies that once a name is specified, the named objects can be accessed unambiguously without additional specification.
- **Replication transparency.** As we show in Figure 25.2, copies of data may be stored at multiple sites for better availability, performance, and reliability. Replication transparency makes the user unaware of the existence of copies.
- **Fragmentation transparency.** Two types of fragmentation are possible. **Horizontal fragmentation** distributes a relation into sets of tuples (rows).

Figure 25.2
Data distribution and replication among distributed databases.



Vertical fragmentation distributes a relation into subrelations where each subrelation is defined by a subset of the columns of the original relation. A global query by the user must be transformed into several fragment queries. Fragmentation transparency makes the user unaware of the existence of fragments.

- Other transparencies include **design transparency** and **execution transparency**—referring to freedom from knowing how the distributed database is designed and where a transaction executes.

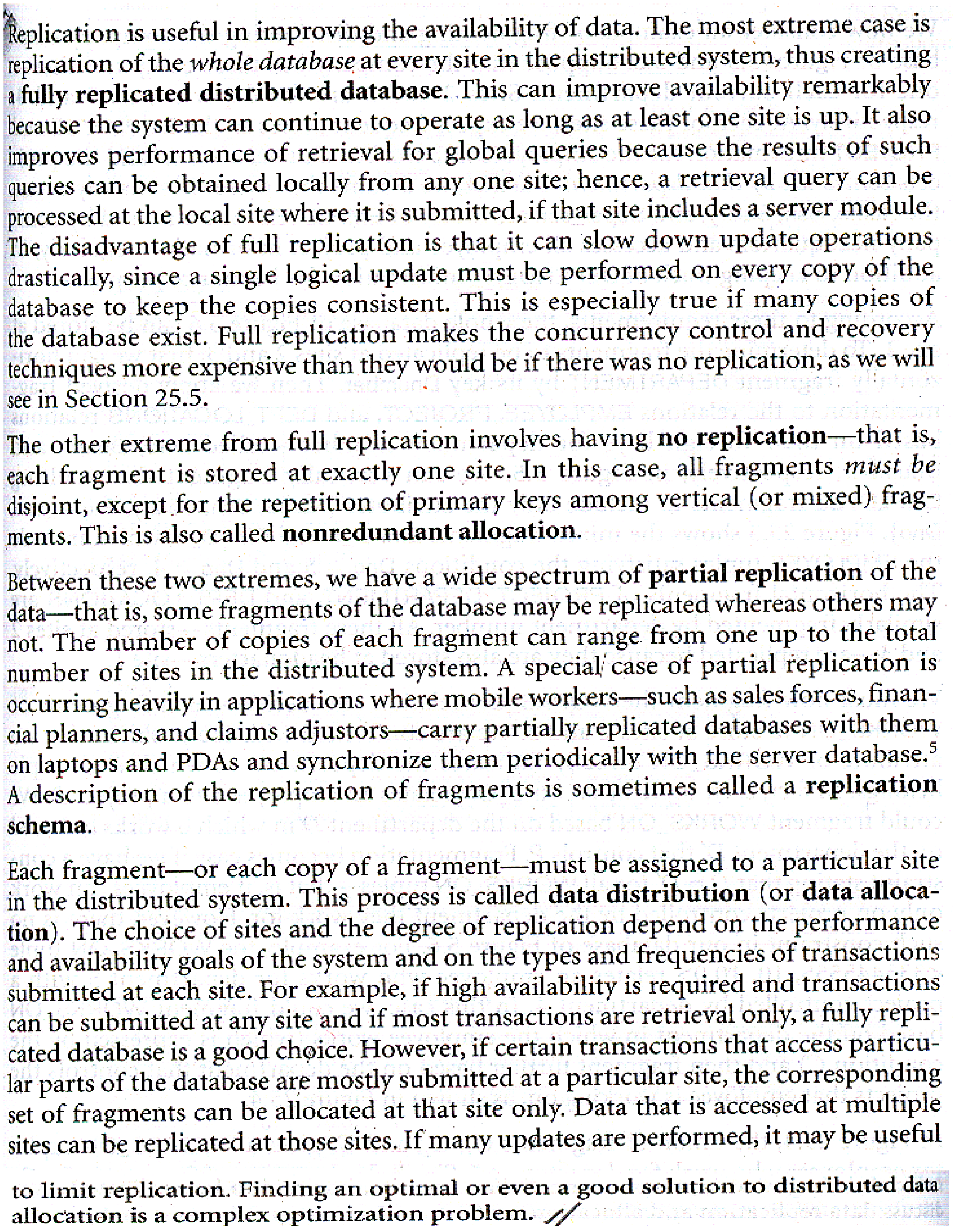
2. **Increased reliability and availability.** These are two of the most common potential advantages cited for distributed databases. **Reliability** is broadly defined as the probability that a system is running (not down) at a certain time point, whereas **availability** is the probability that the system is continuously available during a time interval. When the data and DBMS software are distributed over several sites, one site may fail while other sites continue to operate. Only the data and software that exist at the failed site cannot be accessed. This improves both reliability and availability. Further improvement is achieved by judiciously *replicating* data and software at more than one site. In a centralized system, failure at a single site makes the whole system unavailable to all users. In a distributed database, some of the data may be unreachable, but users may still be able to access other parts of the database.

3. **Improved performance.** A distributed DBMS fragments the database by keeping the data closer to where it is needed most. **Data localization** reduces the contention for CPU and I/O services and simultaneously reduces access delays involved in wide area networks. When a large database is distributed over multiple sites, smaller databases exist at each site. As a result, local queries and transactions accessing data at a single site have better performance because of the smaller local databases. In addition, each site has a smaller number of transactions executing than if all transactions are submitted to a single centralized database. Moreover, interquery and intraquery parallelism can be achieved by executing multiple queries at different sites, or by breaking up a query into a number of subqueries that execute in parallel. This contributes to improved performance.

4. **Easier expansion.** In a distributed environment, expansion of the system in terms of adding more data, increasing database sizes, or adding more processors is much easier.

c. Why is data replication useful in distributed DBMS? (5)

Answer:



Replication is useful in improving the availability of data. The most extreme case is replication of the *whole database* at every site in the distributed system, thus creating a **fully replicated distributed database**. This can improve availability remarkably because the system can continue to operate as long as at least one site is up. It also improves performance of retrieval for global queries because the results of such queries can be obtained locally from any one site; hence, a retrieval query can be processed at the local site where it is submitted, if that site includes a server module. The disadvantage of full replication is that it can slow down update operations drastically, since a single logical update must be performed on every copy of the database to keep the copies consistent. This is especially true if many copies of the database exist. Full replication makes the concurrency control and recovery techniques more expensive than they would be if there was no replication, as we will see in Section 25.5.

The other extreme from full replication involves having **no replication**—that is, each fragment is stored at exactly one site. In this case, all fragments *must be* disjoint, except for the repetition of primary keys among vertical (or mixed) fragments. This is also called **nonredundant allocation**.

Between these two extremes, we have a wide spectrum of **partial replication** of the data—that is, some fragments of the database may be replicated whereas others may not. The number of copies of each fragment can range from one up to the total number of sites in the distributed system. A special case of partial replication is occurring heavily in applications where mobile workers—such as sales forces, financial planners, and claims adjustors—carry partially replicated databases with them on laptops and PDAs and synchronize them periodically with the server database.[5] A description of the replication of fragments is sometimes called a **replication schema**.

Each fragment—or each copy of a fragment—must be assigned to a particular site in the distributed system. This process is called **data distribution** (or **data allocation**). The choice of sites and the degree of replication depend on the performance and availability goals of the system and on the types and frequencies of transactions submitted at each site. For example, if high availability is required and transactions can be submitted at any site and if most transactions are retrieval only, a fully replicated database is a good choice. However, if certain transactions that access particular parts of the database are mostly submitted at a particular site, the corresponding set of fragments can be allocated at that site only. Data that is accessed at multiple sites can be replicated at those sites. If many updates are performed, it may be useful to limit replication. Finding an optimal or even a good solution to distributed data allocation is a complex optimization problem.

Q.9 (For Current Scheme student i.e. AC61/AT61)

a. Discuss the concept of serializability. What is a conflict & View Serializable schedule? (6)

Answer: DBMS must control concurrent execution of transactions to ensure read consistency, i.e., to avoid dirty reads etc.
A (possibly concurrent) schedule S is serializable if it is equivalent to a serial schedule S', i.e., S has the same result database state as S'.

Conflict-serializability is defined by equivalence to a serial schedule (no overlapping transactions) with the same transactions, such that both schedules have the same sets of respective chronologically ordered pairs of conflicting operations (same precedence relations of respective conflicting operations).

View-serializability of a schedule is defined by equivalence to a serial schedule (no overlapping transactions) with the same transactions, such that respective transactions in the two schedules read and write the same data values ("view" the same data values).

b. What is the Purpose of a 2 Phase Commit protocol? How does it work?(6)

Answer: A commit operation is, by definition, an all-or-nothing affair. If a series of operations bound as a transaction cannot be completed, the rollback must restore the system (or cooperating systems) to the pre-transaction state.

In order to ensure that a transaction can be rolled back, a software system typically logs each operation, including the commit operation itself. A transaction/recovery manager uses the log records to undo (and possibly redo) a partially completed transaction.

The commit process proceeds as follows:

Phase 1

- Each participating resource manager coordinates local operations and forces all log records out:
- If successful, respond "OK"
- If unsuccessful, either allow a time-out or respond "OOPS"

Phase 2

- If all participants respond "OK":
- Coordinator instructs participating resource managers to "COMMIT"
  Participants complete operation writing the log record for the commit

Otherwise:

- Coordinator instructs participating resource managers to "ROLLBACK"
  - Participants complete their respective local undos

In order for the scheme to work reliably, both the coordinator and the participating resource managers independently must be able to guarantee proper completion, including any necessary restart/redo operations. The algorithms for guaranteeing success by handling failures at any stage are provided in advanced database texts.

c. Define the followings: (4)
i) Multiple granularity
ii) Intention lock mode

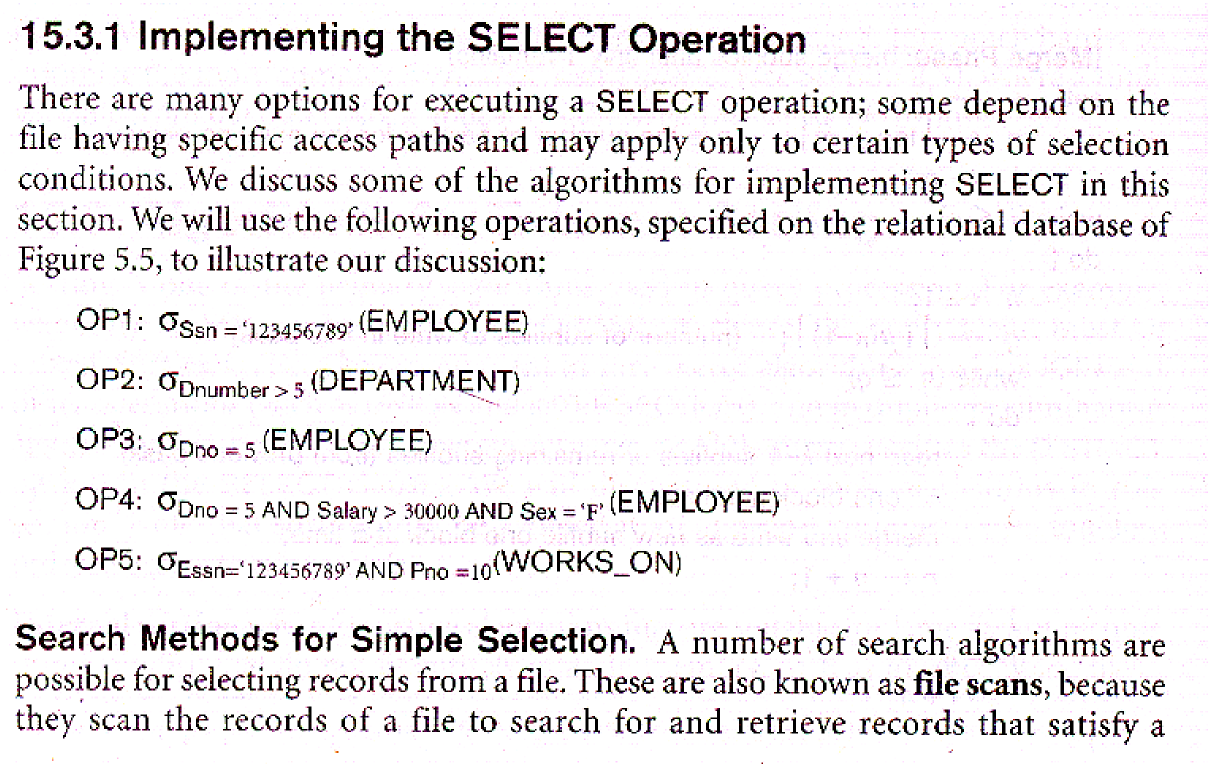
Answer: i) **Multiple Granularity**
Allow data items to be of various sizes and define a hierarchy of data granularities, where the small granularities are nested within larger ones
■ Can be represented graphically as a tree (but don't confuse with tree-locking protocol)
■When a transaction locks a node in the tree explicitly, it implicitly locks all the node's descendents in the same mode.
■ Granularity of locking (level in tree where locking is done):
● fine granularity (lower in tree): high concurrency, high locking overhead
● coarse granularity (higher in tree): low locking overhead, low concurrency

ii) **Intention lock mode**
In addition to S and X lock modes, there are three additional lock modes with multiple granularity:
●intention-shared (IS): indicates explicit locking at a lower level of the tree but only with shared locks.
●intention-exclusive (IX): indicates explicit locking at a lower level with exclusive or shared locks

●shared and intention-exclusive (SIX): the subtree rooted by that node is locked explicitly in shared mode and explicit locking is being done at a lower level with exclusive-mode locks.
■intention locks allow a higher level node to be locked in S or X mode without having to check all descendent nodes.

Q.9 (For New Scheme student i.e. AC112/AT112)
a. Discuss the algorithm for SELECT and JOIN operations. (8)

Answer:



15.3.1 Implementing the SELECT Operation

There are many options for executing a SELECT operation; some depend on the file having specific access paths and may apply only to certain types of selection conditions. We discuss some of the algorithms for implementing SELECT in this section. We will use the following operations, specified on the relational database of Figure 5.5, to illustrate our discussion:

OP1: $\sigma_{\text{Ssn} = \text{'123456789'}}(\text{EMPLOYEE})$

OP2: $\sigma_{\text{Dnumber} > 5}(\text{DEPARTMENT})$

OP3: $\sigma_{\text{Dno} = 5}(\text{EMPLOYEE})$

OP4: $\sigma_{\text{Dno} = 5 \text{ AND Salary} > 30000 \text{ AND Sex} = \text{'F'}}(\text{EMPLOYEE})$

OP5: $\sigma_{\text{Essn}=\text{'123456789'} \text{ AND Pno} = 10}(\text{WORKS\_ON})$

Search Methods for Simple Selection. A number of search algorithms are possible for selecting records from a file. These are also known as **file scans**, because they scan the records of a file to search for and retrieve records that satisfy a selection condition.[4] If the search algorithm involves the use of an index, the index search is called an **index scan**. The following search methods (S1 through S6) are examples of some of the search algorithms that can be used to implement a select operation:

- **S1—Linear search (brute force).** Retrieve *every record* in the file, and test whether its attribute values satisfy the selection condition.
- **S2—Binary search.** If the selection condition involves an equality comparison on a key attribute on which the file is ordered, binary search—which is more efficient than linear search—can be used. An example is OP1 if Ssn is the ordering attribute for the EMPLOYEE file.[5]
- **S3—Using a primary index (or hash key).** If the selection condition involves an equality comparison on a **key attribute** with a primary index (or hash key)—for example, Ssn = '123456789' in OP1—use the primary index (or hash key) to retrieve the record. Note that this condition retrieves a single record (at most).
- **S4—Using a primary index to retrieve multiple records.** If the comparison condition is >, >=, <, or <= on a key field with a primary index—for example, Dnumber > 5 in OP2—use the index to find the record satisfying the corresponding equality condition (Dnumber = 5), then retrieve all subsequent records in the (ordered) file. For the condition Dnumber < 5, retrieve all the preceding records.
- **S5—Using a clustering index to retrieve multiple records.** If the selection condition involves an equality comparison on a **nonkey attribute** with a clustering index—for example, Dno = 5 in OP3—use the index to retrieve all the records satisfying the condition.
- **S6—Using a secondary (B^+-tree) index on an equality comparison.** This search method can be used to retrieve a single record if the indexing field is a **key** (has unique values) or to retrieve multiple records if the indexing field is **not a key**. This can also be used for comparisons involving >, >=, <, or <=.

In Section 15.8, we discuss how to develop formulas that estimate the access cost of these search methods in terms of number of block accesses and access time. Method S1 applies to any file, but all the other methods depend on having the appropriate access path on the attribute used in the selection condition. Methods S4 and S6 can be used to retrieve records in a certain *range*—for example, 30000 <= Salary <= 35000. Queries involving such conditions are called **range queries**.

Search Methods for Complex Selection. If a condition of a SELECT operation is a **conjunctive condition**—that is, if it is made up of several simple conditions

4. A selection operation is sometimes called a **filter**, since it filters out the records in the file that do *not* satisfy the selection condition.

5. Generally, binary search is not used in database search because ordered files are not used unless they also have a corresponding primary index.

connected with the AND logical connective such as OP4 above—the DBMS can use the following additional methods to implement the operation:

- **S7—Conjunctive selection using an individual index.** If an attribute involved in any **single simple condition** in the conjunctive condition has an access path that permits the use of one of the Methods S2 to S6, use that condition to retrieve the records and then check whether each retrieved record *satisfies the remaining simple conditions* in the conjunctive condition.
- **S8—Conjunctive selection using a composite index.** If two or more attributes are involved in equality conditions in the conjunctive condition and a composite index (or hash structure) exists on the combined fields—for example, if an index has been created on the composite key (Essn, Pno) of the WORKS_ON file for OP5—we can use the index directly.
- **S9—Conjunctive selection by intersection of record pointers.**[6] If secondary indexes (or other access paths) are available on more than one of the fields involved in simple conditions in the conjunctive condition, and if the indexes include record pointers (rather than block pointers), then each index can be used to retrieve the **set of record pointers** that satisfy the individual condition. The **intersection** of these sets of record pointers gives the record pointers that satisfy the conjunctive condition, which are then used to retrieve those records directly. If only some of the conditions have secondary indexes, each retrieved record is further tested to determine whether it satisfies the remaining conditions.[7]

Whenever a single condition specifies the selection—such as OP1, OP2, or OP3—we can only check whether an access path exists on the attribute involved in that condition. If an access path exists, the method corresponding to that access path is used; otherwise, the brute force linear search approach of method S1 can be used. Query optimization for a SELECT operation is needed mostly for conjunctive select conditions whenever *more than one* of the attributes involved in the conditions have an access path. The optimizer should choose the access path that *retrieves the fewest records* in the most efficient way by estimating the different costs (see Section 15.8) and choosing the method with the least estimated cost.

When the optimizer is choosing between multiple simple conditions in a conjunctive select condition, it typically considers the selectivity of each condition. The **selectivity** (*s*) is defined as the ratio of the number of records (tuples) that satisfy the condition to the total number of records (tuples) in the file (relation), and thus is a number between zero and 1—zero selectivity means no records satisfy the condition and 1 means all the records satisfy the condition. Although exact selectivities of all conditions may not be available, **estimates of selectivities** are often kept in the DBMS catalog and are used by the optimizer. For example, for an equality condition

6. A record pointer uniquely identifies a record and provides the address of the record on disk; hence, it is also called the **record identifier** or **record id**.

7. The technique can have many variations—for example, if the indexes are *logical indexes* that store primary key values instead of record pointers.

on a key attribute of relation $r(R)$, $s = 1/|r(R)|$, where $|r(R)|$ is the number of tuples in relation $r(R)$. For an equality condition on an attribute with *i distinct values*, s can be estimated by $(|r(R)|/i)/|r(R)|$ or $1/i$, assuming that the records are evenly distributed among the distinct values.[8] Under this assumption, $|r(R)|/i$ records will satisfy an equality condition on this attribute. In general, the number of records satisfying a selection condition with selectivity s is estimated to be $|r(R)| * s$. The smaller this estimate is, the higher the desirability of using that condition first to retrieve records.

Compared to a conjunctive selection condition, a **disjunctive condition** (where simple conditions are connected by the OR logical connective rather than by AND) is much harder to process and optimize. For example, consider OP4′:

OP4′: $\sigma_{\text{Dno}=5 \text{ OR Salary}>30000 \text{ OR Sex}=\text{'F'}}$ (EMPLOYEE)

With such a condition, little optimization can be done, because the records satisfying the disjunctive condition are the *union* of the records satisfying the individual conditions. Hence, if any *one* of the conditions does not have an access path, we are compelled to use the brute force linear search approach. Only if an access path exists on *every* condition can we optimize the selection by retrieving the records satisfying each condition—or their record ids—and then applying the union operation to eliminate duplicates.

A DBMS will have available many of the methods discussed above, and typically many additional methods. The query optimizer must choose the appropriate one for executing each SELECT operation in a query. This optimization uses formulas that estimate the costs for each available access method, as we shall discuss in Section 15.8. The optimizer chooses the access method with the lowest estimated cost.

15.3.2 Implementing the JOIN Operation

The JOIN operation is one of the most time-consuming operations in query processing. Many of the join operations encountered in queries are of the EQUIJOIN and NATURAL JOIN varieties, so we consider only these two here. For the remainder of this chapter, the term **join** refers to an EQUIJOIN (or NATURAL JOIN). There are many possible ways to implement a **two-way join**, which is a join on two files. Joins involving more than two files are called **multiway joins**. The number of possible ways to execute multiway joins grows very rapidly. In this section we discuss techniques for implementing only two-way joins. To illustrate our discussion, we refer to the relational schema of Figure 5.5 once more—specifically, to the EMPLOYEE, DEPARTMENT, and PROJECT relations. The algorithms we consider are for join operations of the form

$$R \bowtie_{A=B} S$$

where A and B are domain-compatible attributes of R and S, respectively. The methods we discuss can be extended to more general forms of join. We illustrate four of

8. In more sophisticated optimizers, histograms representing the distribution of the records among the different attribute values can be kept in the catalog.

the most common techniques for performing such a join, using the following example operations:

OP6: EMPLOYEE $\bowtie_{\text{Dno=Dnumber}}$ DEPARTMENT
OP7: DEPARTMENT $\bowtie_{\text{Mgr\_ssn=Ssn}}$ EMPLOYEE

Methods for Implementing Joins.

- **J1—Nested-loop join (brute force).** For each record *t* in *R* (outer loop), retrieve every record *s* from *S* (inner loop) and test whether the two records satisfy the join condition $t[A] = s[B]$.[9]
- **J2—Single-loop join (using an access structure to retrieve the matching records).** If an index (or hash key) exists for one of the two join attributes—say, *B* of *S*—retrieve each record *t* in *R*, one at a time (single loop), and then use the access structure to retrieve directly all matching records *s* from *S* that satisfy $s[B] = t[A]$.
- **J3—Sort-merge join.** If the records of *R* and *S* are *physically sorted* (ordered) by value of the join attributes *A* and *B*, respectively, we can implement the join in the most efficient way possible. Both files are scanned concurrently in order of the join attributes, matching the records that have the same values for *A* and *B*. If the files are not sorted, they may be sorted first by using external sorting (see Section 15.2). In this method, pairs of file blocks are copied into memory buffers in order and the records of each file are scanned only once each for matching with the other file—unless both *A* and *B* are nonkey attributes, in which case the method needs to be modified slightly. A sketch of the sort-merge join algorithm is given in Figure 15.3(a). We use $R(i)$ to refer to the *i*th record in *R*. A variation of the sort-merge join can be used when secondary indexes exist on both join attributes. The indexes provide the ability to access (scan) the records in order of the join attributes, but the records themselves are physically scattered all over the file blocks, so this method may be quite inefficient, as every record access may involve accessing a different disk block.
- **J4—Hash-join.** The records of files *R* and *S* are both hashed to the same hash file, using the same hashing function on the join attributes *A* of *R* and *B* of *S* as hash keys. First, a single pass through the file with fewer records (say, *R*) hashes its records to the hash file buckets; this is called the **partitioning phase**, since the records of *R* are partitioned into the hash buckets. In the second phase, called the **probing phase**, a single pass through the other file (*S*) then hashes each of its records to *probe* the appropriate bucket, and that record is combined with all matching records from *R* in that bucket. This simplified description of hash-join assumes that the smaller of the two files *fits entirely into memory buckets* after the first phase. We will discuss variations of hash-join that do not require this assumption below.

9. For disk files, it is obvious that the loops will be over disk blocks so this technique has also been called *nested-block join*.

```
(a) sort the tuples in $R$ on attribute $A$;    (* assume $R$ has $n$ tuples (records) *)
    sort the tuples in $S$ on attribute $B$;    (* assume $S$ has $m$ tuples (records) *)
    set $i \leftarrow 1, j \leftarrow 1$;
    while $(i \leq n)$ and $(j \leq m)$
    do {  if $R(i)[A] > S(j)[B]$
             then  set $j \leftarrow j + 1$
          elseif $R(i)[A] < S(j)[B]$
             then  set $i \leftarrow i + 1$
          else  {  (* $R(i)[A] = S(j)[B]$, so we output a matched tuple *)
                   output the combined tuple $<R(i), S(j)>$ to $T$;

                   (* output other tuples that match $R(i)$, if any *)
                   set $l \leftarrow j + 1$;
                   while $(l \leq m)$ and $(R(i)[A] = S(l)[B])$
                   do {  output the combined tuple $<R(i), S(l)>$ to $T$;
                         set $l \leftarrow l + 1$
                   }

                   (* output other tuples that match $S(j)$, if any *)
                   set $k \leftarrow i + 1$;
                   while $(k \leq n)$ and $(R(k)[A] = S(j)[B])$
                   do {  output the combined tuple $<R(k), S(j)>$ to $T$;
                         set $k \leftarrow k + 1$
                   }
                   set $i \leftarrow k, j \leftarrow l$
          }
    }

(b) create a tuple $t$[<attribute list>] in $T'$ for each tuple $t$ in R;
        (* $T'$ contains the projection results before duplicate elimination *)
    if <attribute list> includes a key of $R$
         then $T \leftarrow T'$
    else  {  sort the tuples in $T'$;
             set $i \leftarrow 1, j \leftarrow 2$;
             while $i \leq n$
             do {  output the tuple $T'[i]$ to $T$;
                   while $T'[i] = T'[j]$ and $j \leq n$ do $j \leftarrow j + 1$;   (* eliminate duplicates *)
                   $i \leftarrow j; j \leftarrow i + 1$
             }
    }
    (* $T$ contains the projection result after duplicate elimination *)
```

Figure 15.3
Implementing JOIN, PROJECT, UNION, INTERSECTION, and SET DIFFERENCE by using sort-merge, where R has n tuples and S has m tuples. (a) Implementing the operation $T \leftarrow R \quad {}_{A=B}S$. (b) Implementing the operation $T \leftarrow \pi_{\text{<attribute list>}}(R)$.

(c)
```
sort the tuples in $R$ and $S$ using the same unique sort attributes;
set $i \leftarrow 1, j \leftarrow 1$;
while $(i \leq n)$ and $(j \leq m)$
do {  if $R(i) > S(j)$
          then  {  output $S(j)$ to $T$;
                   set $j \leftarrow j + 1$
          }
      elseif $R(i) < S(j)$
          then  {  output $R(i)$ to $T$;
                   set $i \leftarrow i + 1$
          }
      else set $j \leftarrow j + 1$   (* $R(i)=S(j)$, so we skip one of the duplicate tuples *)
}
if $(i \leq n)$ then add tuples $R(i)$ to $R(n)$ to $T$;
if $(j \leq m)$ then add tuples $S(j)$ to $S(m)$ to $T$;
```

(d)
```
sort the tuples in $R$ and $S$ using the same unique sort attributes;
set $i \leftarrow 1, j \leftarrow 1$;
while $(i \leq n)$ and $(j \leq m)$
do {  if $R(i) > S(j)$
          then  set $j \leftarrow j + 1$
      elseif $R(i) < S(j)$
          then  set $i \leftarrow i + 1$
      else  {  output $R(j)$ to $T$;  (* $R(i)=S(j)$, so we output the tuple *)
               set $i \leftarrow i + 1, j \leftarrow j + 1$
      }
}
```

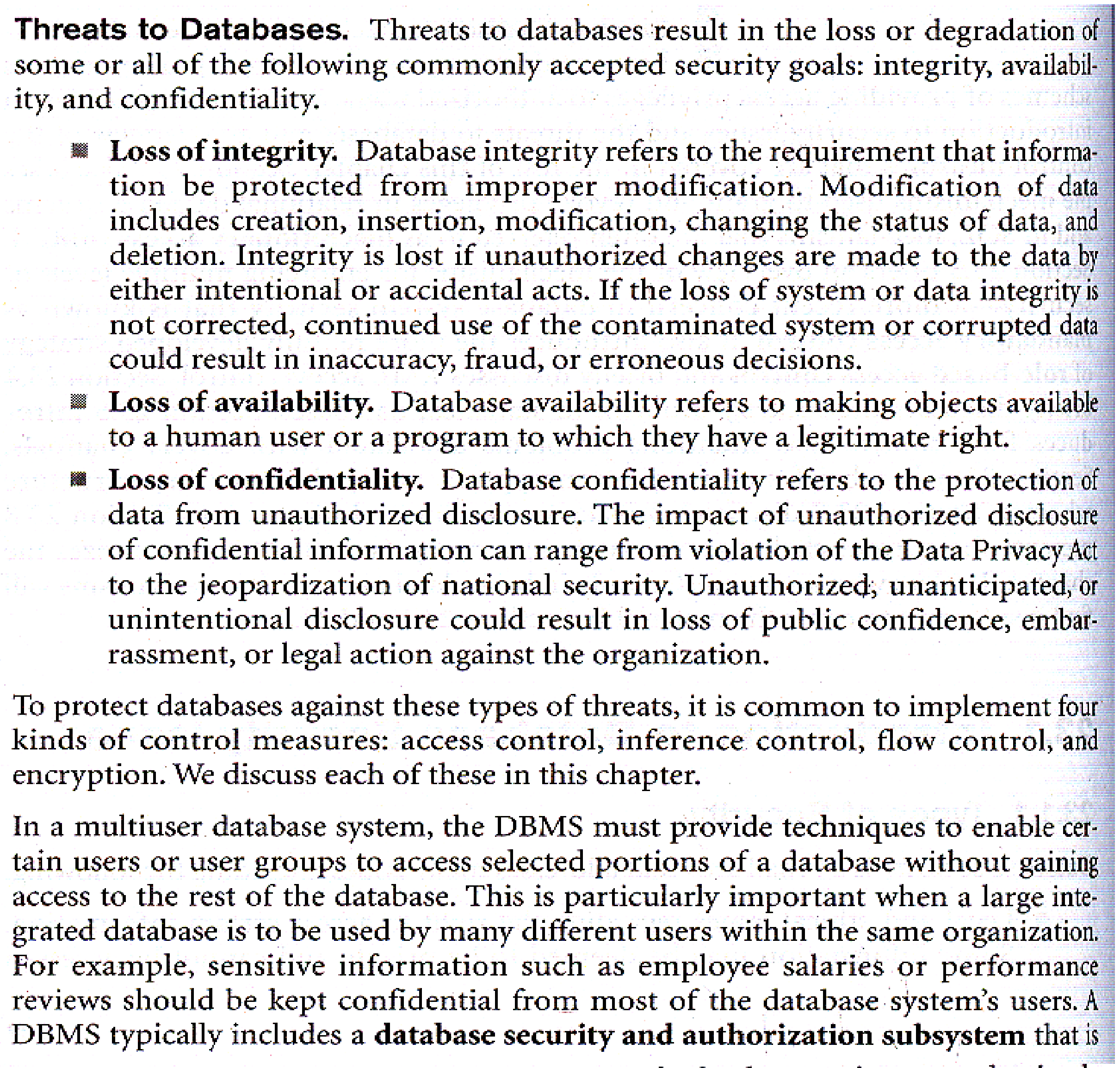
(e)
```
sort the tuples in $R$ and $S$ using the same unique sort attributes;
set $i \leftarrow 1, j \leftarrow 1$;
while $(i \leq n)$ and $(j \leq m)$
do {  if $R(i) > S(j)$
          then  set $j \leftarrow j + 1$
      elseif $R(i) < S(j)$
          then  {  output $R(i)$ to $T$;  (* $R(i)$ has no matching $S(j)$, so output $R(i)$ *)
                   set $i \leftarrow i + 1$
          }
      else  set $i \leftarrow i + 1, j \leftarrow j + 1$
}
if $(i \leq n)$ then add tuples $R(i)$ to $R(n)$ to $T$;
```

Figure 15.3 (continued)
Implementing JOIN, PROJECT, UNION, INTERSECTION, and SET DIFFERENCE by using sort-merge, where R has n tuples and S has m tuples. (c) Implementing the operation $T \leftarrow R \cup S$. (d) Implementing the operation $T \leftarrow R \cap S$. (e) Implementing the operation $T \leftarrow R - S$.

In practice, techniques J1 to J4 are implemented by accessing *whole disk blocks* of a file, rather than individual records. Depending on the available buffer space in memory, the number of blocks read in from the file can be adjusted.

b. List the commonly accepted threats to database security. (4)

Answer:

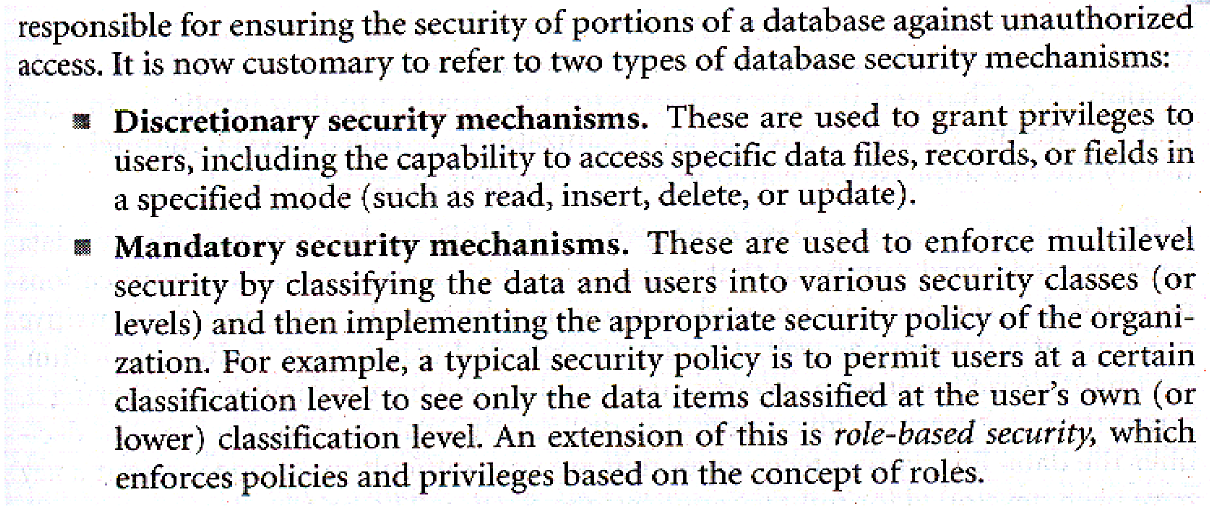


Threats to Databases. Threats to databases result in the loss or degradation of some or all of the following commonly accepted security goals: integrity, availability, and confidentiality.

- **Loss of integrity.** Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creation, insertion, modification, changing the status of data, and deletion. Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decisions.
- **Loss of availability.** Database availability refers to making objects available to a human user or a program to which they have a legitimate right.
- **Loss of confidentiality.** Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to the jeopardization of national security. Unauthorized, unanticipated, or unintentional disclosure could result in loss of public confidence, embarrassment, or legal action against the organization.

To protect databases against these types of threats, it is common to implement four kinds of control measures: access control, inference control, flow control, and encryption. We discuss each of these in this chapter.

In a multiuser database system, the DBMS must provide techniques to enable certain users or user groups to access selected portions of a database without gaining access to the rest of the database. This is particularly important when a large integrated database is to be used by many different users within the same organization. For example, sensitive information such as employee salaries or performance reviews should be kept confidential from most of the database system's users. A DBMS typically includes a **database security and authorization subsystem** that is

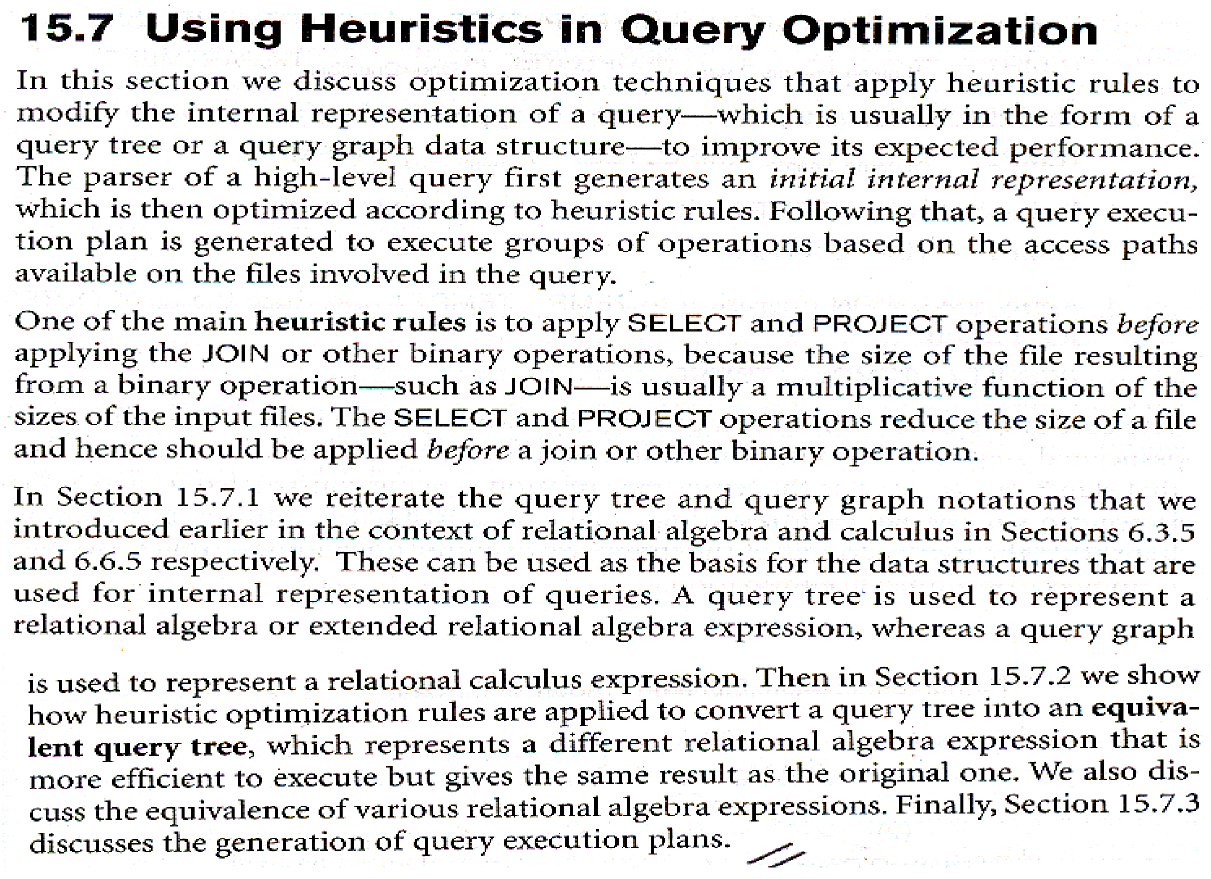


responsible for ensuring the security of portions of a database against unauthorized access. It is now customary to refer to two types of database security mechanisms:

- **Discretionary security mechanisms.** These are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).
- **Mandatory security mechanisms.** These are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization. For example, a typical security policy is to permit users at a certain classification level to see only the data items classified at the user's own (or lower) classification level. An extension of this is *role-based security,* which enforces policies and privileges based on the concept of roles.

c. What is meant by the term heuristic optimization, discuss. (4)

Answer:



15.7 Using Heuristics in Query Optimization

In this section we discuss optimization techniques that apply heuristic rules to modify the internal representation of a query—which is usually in the form of a query tree or a query graph data structure—to improve its expected performance. The parser of a high-level query first generates an *initial internal representation,* which is then optimized according to heuristic rules. Following that, a query execution plan is generated to execute groups of operations based on the access paths available on the files involved in the query.

One of the main **heuristic rules** is to apply SELECT and PROJECT operations *before* applying the JOIN or other binary operations, because the size of the file resulting from a binary operation—such as JOIN—is usually a multiplicative function of the sizes of the input files. The SELECT and PROJECT operations reduce the size of a file and hence should be applied *before* a join or other binary operation.

In Section 15.7.1 we reiterate the query tree and query graph notations that we introduced earlier in the context of relational algebra and calculus in Sections 6.3.5 and 6.6.5 respectively. These can be used as the basis for the data structures that are used for internal representation of queries. A query tree is used to represent a relational algebra or extended relational algebra expression, whereas a query graph is used to represent a relational calculus expression. Then in Section 15.7.2 we show how heuristic optimization rules are applied to convert a query tree into an **equivalent query tree**, which represents a different relational algebra expression that is more efficient to execute but gives the same result as the original one. We also discuss the equivalence of various relational algebra expressions. Finally, Section 15.7.3 discusses the generation of query execution plans.

TEXT BOOK

I. Fundamentals of Database Systems, Elmasri, Navathe, Somayajulu, Gupta, Pearson Education, 2006.