CHAPTER 18

Parallel Databases

Practice Exercises

18.1 In a range selection on a range-partitioned attribute, it is possible that only one disk may need to be accessed. Describe the benefits and drawbacks of this property.

Answer: If there are few tuples in the queried range, then each query can be processed quickly on a single disk. This allows parallel execution of queries with reduced overhead of initiating queries on multiple disks.

On the other hand, if there are many tuples in the queried range, each query takes a long time to execute as there is no parallelism within its execution. Also, some of the disks can become hot-spots, further increasing response time.

Hybrid range partitioning, in which small ranges (a few blocks each) are partitioned in a round-robin fashion, provides the benefits of range partitioning without its drawbacks.

18.2 What form of parallelism (interquery, interoperation, or intraoperation) is likely to be the most important for each of the following tasks?

a. Increasing the throughput of a system with many small queries

b. Increasing the throughput of a system with a few large queries, when the number of disks and processors is large

Answer:

a. When there are many small queries, inter-query parallelism gives good throughput. Parallelizing each of these small queries would increase the initiation overhead, without any significant reduction in response time.

b. With a few large queries, intra-query parallelism is essential to get fast response times. Given that there are large number of processors and disks, only intra-operation parallelism can take advantage of the parallel hardware – for queries typically have

few operations, but each one needs to process a large number of tuples.

18.3 With pipelined parallelism, it is often a good idea to perform several operations in a pipeline on a single processor, even when many processors are available.

a. Explain why.

b. Would the arguments you advanced in part *a* hold if the machine has a shared-memory architecture? Explain why or why not.

c. Would the arguments in part *a* hold with independent parallelism? (That is, are there cases where, even if the operations are not pipelined and there are many processors available, it is still a good idea to perform several operations on the same processor?)

Answer:

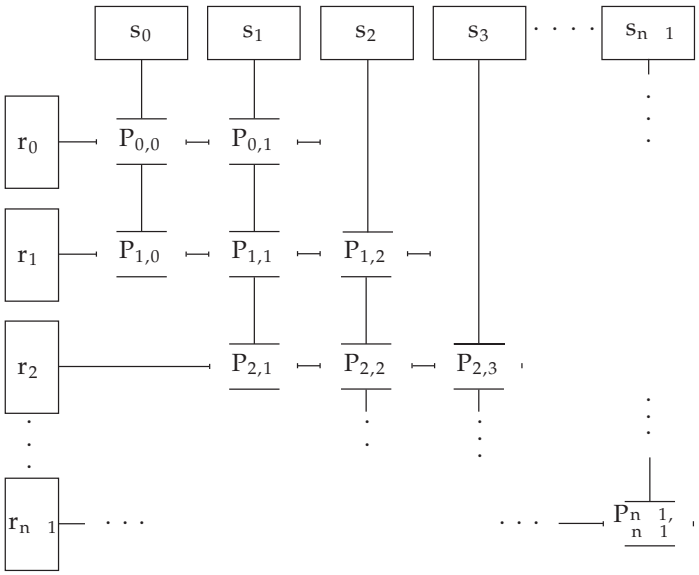
a. The speed-up obtained by parallelizing the operations would be offset by the data transfer overhead, as each tuple produced by an operator would have to be transferred to its consumer, which is running on a different processor.

b. In a shared-memory architecture, transferring the tuples is very efficient. So the above argument does not hold to any significant degree.

c. Even if two operations are independent, it may be that they both supply their outputs to a common third operator. In that case, running all three on the same processor may be better than transferring tuples across processors.

18.4 Consider join processing using symmetric fragment and replicate with range partitioning. How can you optimize the evaluation if the join condition is of the form $| r.A - s.B | \leq k$, where k is a small constant? Here, $| x |$ denotes the absolute value of x. A join with such a join condition is called a **band join**.

Answer: Relation r is partitioned into n partitions, $r_0, r_1, \ldots, r_{n-1}$, and s is also partitioned into n partitions, $s_0, s_1, \ldots, s_{n-1}$. The partitions are replicated and assigned to processors as shown below.



Each fragment is replicated on 3 processors only, unlike in the general case where it is replicated on n processors. The number of processors required is now approximately $3n$, instead of n^2 in the general case. Therefore given the same number of processors, we can partition the relations into more fragments with this optimization, thus making each local join faster.

18.5 Recall that histograms are used for constructing load-balanced range partitions.

a. Suppose you have a histogram where values are between 1 and 100, and are partitioned into 10 ranges, 1–10, 11–20, . . . , 91–100, with frequencies 15, 5, 20, 10, 10, 5, 5, 20, 5, and 5, respectively. Give a load-balanced range partitioning function to divide the values into 5 partitions.

b. Write an algorithm for computing a balanced range partition with p partitions, given a histogram of frequency distributions containing n ranges.

Answer:

a. A partitioning vector which gives 5 partitions with 20 tuples in each partition is: [21, 31, 51, 76]. The 5 partitions obtained are $1-20, 21-30, 31-50, 51-75$ and $76-100$. The assumption made in arriving at this partitioning vector is that within a histogram range, each value is equally likely.

b. Let the histogram ranges be called $h_1, h_2, \ldots, h_h$, and the partitions $p_1, p_2, \ldots, p_p$. Let the frequencies of the histogram ranges be

$n_1, n_2, \ldots, n_h$. Each partition should contain N/p tuples, where $N = \Sigma_{i=1}^{h} n_i$.
To construct the load balanced partitioning vector, we need to determine the value of the k_1^{th} tuple, the value of the k_2^{th} tuple and so on, where $k_1 = N/p, k_2 = 2N/p$ etc, until k_{p-1}. The partitioning vector will then be $[k_1, k_2, \ldots, k_{p-1}]$. The value of the k_i^{th} tuple is determined as follows. First determine the histogram range h_j in which it falls. Assuming all values in a range are equally likely, the k_i^{th} value will be

$$s_j + (e_j - s_j) * \frac{k_{ij}}{n_j}$$

where

s_j	:	first value in h_j
e_j	:	last value in h_j
k_{ij}	:	$k_i - \Sigma_{l=1}^{j-1} n_l$

18.6 Large-scale parallel database systems store an extra copy of each data item on disks attached to a different processor, to avoid loss of data if one of the processors fails.

a. Instead of keeping the extra copy of data items from a processor at a single backup processor, it is a good idea to partition the copies of the data items of a processor across multiple processors. Explain why.

b. Explain how virtual-processor partitioning can be used to efficiently implement the partitioning of the copies as described above.

c. What are the benefits and drawbacks of using RAID storage instead of storing an extra copy of each data item?

Answer: FILL

18.7 Suppose we wish to index a large relation that is partitioned. Can the idea of partitioning (including virtual processor partitioning) be applied to indices? Explain your answer, considering the following two cases (assuming for simplicity that partitioning as well as indexing are on single attributes):

a. Where the index is on the partitioning attribute of the relation.

b. Where the index is on an attribute other than the partitioning attribute of the relation.

Answer: FILL

18.8 Suppose a well-balanced range-partitioning vector had been chosen for a relation, but the relation is subsequently updated, making the partitioning unbalanced. Even if virtual-processor partitioning is used,

a particular virtual processor may end up with a very large number of tuples after the update, and repartitioning would then be required.

a. Suppose a virtual processor has a significant excess of tuples (say, twice the average). Explain how repartitioning can be done by splitting the partition, thereby increasing the number of virtual processors.

b. If, instead of round-robin allocation of virtual processors, virtual partitions can be allocated to processors in an arbitrary fashion, with a mapping table tracking the allocation. If a particular node has excess load (compared to the others), explain how load can be balanced.

c. Assuming there are no updates, does query processing have to be stopped while repartitioning, or reallocation of virtual processors, is carried out? Explain your answer.

Answer: FILL

CHAPTER 19

Distributed Databases

Practice Exercises

19.1 How might a distributed database designed for a local-area network differ from one designed for a wide-area network?

Answer: Data transfer on a local-area network (LAN) is much faster than on a wide-area network (WAN). Thus replication and fragmentation will not increase throughput and speed-up on a LAN, as much as in a WAN. But even in a LAN, replication has its uses in increasing reliability and availability.

19.2 To build a highly available distributed system, you must know what kinds of failures can occur.

a. List possible types of failure in a distributed system.

b. Which items in your list from part a are also applicable to a centralized system?

Answer:

a. The types of failure that can occur in a distributed system include
   i. Site failure.
   ii. Disk failure.
   iii. Communication failure, leading to disconnection of one or more sites from the network.

b. The first two failure types can also occur on centralized systems.

19.3 Consider a failure that occurs during 2PC for a transaction. For each possible failure that you listed in Practice Exercise 19.2a, explain how 2PC ensures transaction atomicity despite the failure.

Answer: A proof that 2PC guarantees atomic commits/aborts inspite of site and link failures, follows. The main idea is that after all sites reply with a <**ready** T> message, only the co-ordinator of a transaction can make a commit or abort decision. Any subsequent commit or abort by a

site can happen only after it ascertains the co-ordinator's decision, either directly from the co-ordinator, or indirectly from some other site. Let us enumerate the cases for a site aborting, and then for a site committing.

a. A site can abort a transaction T (by writing an <**abort** T> log record) only under the following circumstances:

   i. It has not yet written a <**ready** T> log-record. In this case, the co-ordinator could not have got, and will not get a <**ready** T> or <**commit** T> message from this site. Therefore only an abort decision can be made by the co-ordinator.

   ii. It has written the <**ready** T> log record, but on inquiry it found out that some other site has an <**abort** T> log record. In this case it is correct for it to abort, because that other site would have ascertained the co-ordinator's decision (either directly or indirectly) before actually aborting.

   iii. It is itself the co-ordinator. In this case also no site could have committed, or will commit in the future, because commit decisions can be made only by the co-ordinator.

b. A site can commit a transaction T (by writing an <**commit** T> log record) only under the following circumstances:

   i. It has written the <**ready** T> log record, and on inquiry it found out that some other site has a <**commit** T> log record. In this case it is correct for it to commit, because that other site would have ascertained the co-ordinator's decision (either directly or indirectly) before actually committing.

   ii. It is itself the co-ordinator. In this case no other participating site can abort/ would have aborted, because abort decisions are made only by the co-ordinator.

19.4 Consider a distributed system with two sites, A and B. Can site A distinguish among the following?

- B goes down.
- The link between A and B goes down.
- B is extremely overloaded and response time is 100 times longer than normal.

What implications does your answer have for recovery in distributed systems?

Answer:

Site A cannot distinguish between the three cases until communication has resumed with site B. The action which it performs while B is inaccessible must be correct irrespective of which of these situations has actually

occurred, and must be such that B can re-integrate consistently into the distributed system once communication is restored.

19.5 The persistent messaging scheme described in this chapter depends on timestamps combined with discarding of received messages if they are too old. Suggest an alternative scheme based on sequence numbers instead of timestamps.

Answer: We can have a scheme based on sequence numbers similar to the scheme based on timestamps. We tag each message with a sequence number that is unique for the (sending site, receiving site) pair. The number is increased by 1 for each new message sent from the sending site to the receiving site.

The receiving site stores and acknowledges a received message only if it has received all lower numbered messages also; the message is stored in the *received-messages* relation.

The sending site retransmits a message until it has received an ack from the receiving site containing the sequence number of the transmitted message, or a higher sequence number. Once the acknowledgment is received, it can delete the message from its send queue.

The receiving site discards all messages it receives that have a lower sequence number than the latest stored message from the sending site. The receiving site discards from *received-messages* all but the (number of the) most recent message from each sending site (message can be discarded only after being processed locally).

Note that this scheme requires a fixed (and small) overhead at the receiving site for each sending site, regardless of the number of messages received. In contrast the timestamp scheme requires extra space for every message. The timestamp scheme would have lower storage overhead if the number of messages received within the timeout interval is small compared to the number of sites, whereas the sequence number scheme would have lower overhead otherwise.

19.6 Give an example where the read one, write all available approach leads to an erroneous state.

Answer: Consider the balance in an account, replicated at N sites. Let the current balance be \$100 – consistent across all sites. Consider two transactions T_1 and T_2 each depositing \$10 in the account. Thus the balance would be \$120 after both these transactions are executed. Let the transactions execute in sequence: T_1 first and then T_2. Let one of the sites, say s, be down when T_1 is executed and transaction t_2 reads the balance from site s. One can see that the balance at the primary site would be \$110 at the end.

19.7 Explain the difference between data replication in a distributed system and the maintenance of a remote backup site.

Answer: In remote backup systems all transactions are performed at the primary site and the data is replicated at the remote backup site. The

remote backup site is kept synchronized with the updates at the primary site by sending all log records. Whenever the primary site fails, the remote backup site takes over processing.
The distributed systems offer greater availability by having multiple copies of the data at different sites whereas the remote backup systems offer lesser availability at lower cost and execution overhead.
In a distributed system, transaction code runs at all the sites whereas in a remote backup system it runs only at the primary site. The distributed system transactions follow two-phase commit to have the data in consistent state whereas a remote backup system does not follow two-phase commit and avoids related overhead.

19.8 Give an example where lazy replication can lead to an inconsistent database state even when updates get an exclusive lock on the primary (master) copy.

Answer: Consider the balance in an account, replicated at N sites. Let the current balance be \$100 – consistent across all sites. Consider two transactions T_1 and T_2 each depositing \$10 in the account. Thus the balance would be \$120 after both these transactions are executed. Let the transactions execute in sequence: T_1 first and then T_2. Suppose the copy of the balance at one of the sites, say s, is not consistent – due to lazy replication strategy – with the primary copy after transaction T_1 is executed and let transaction T_2 read this copy of the balance. One can see that the balance at the primary site would be \$110 at the end.

19.9 Consider the following deadlock-detection algorithm. When transaction T_i, at site S_1, requests a resource from T_j, at site S_3, a request message with timestamp n is sent. The edge (T_i, T_j, n) is inserted in the local wait-for graph of S_1. The edge (T_i, T_j, n) is inserted in the local wait-for graph of S_3 only if T_j has received the request message and cannot immediately grant the requested resource. A request from T_i to T_j in the same site is handled in the usual manner; no timestamps are associated with the edge (T_i, T_j). A central coordinator invokes the detection algorithm by sending an initiating message to each site in the system.

On receiving this message, a site sends its local wait-for graph to the coordinator. Note that such a graph contains all the local information that the site has about the state of the real graph. The wait-for graph reflects an instantaneous state of the site, but it is not synchronized with respect to any other site.

When the controller has received a reply from each site, it constructs a graph as follows:

- The graph contains a vertex for every transaction in the system.
- The graph has an edge (T_i, T_j) if and only if:
  - There is an edge (T_i, T_j) in one of the wait-for graphs.

- An edge (T_i, T_j, n) (for some n) appears in more than one wait-for graph.

Show that, if there is a cycle in the constructed graph, then the system is in a deadlock state, and that, if there is no cycle in the constructed graph, then the system was not in a deadlock state when the execution of the algorithm began.

Answer: Let us say a cycle $T_i \rightarrow T_j \rightarrow \cdots \rightarrow T_m \rightarrow T_i$ exists in the graph built by the controller. The edges in the graph will either be local edges of the from (T_k, T_l) or distributed edges of the form (T_k, T_l, n). Each local edge (T_k, T_l) definitely implies that T_k is waiting for T_l. Since a distributed edge (T_k, T_l, n) is inserted into the graph only if T_k's request has reached T_l and T_l cannot immediately release the lock, T_k is indeed waiting for T_l. Therefore every edge in the cycle indeed represents a transaction waiting for another. For a detailed proof that this imlies a deadlock refer to Stuart et al. [1984].

We now prove the converse implication. As soon as it is discovered that T_k is waiting for T_l:

a. a local edge (T_k, T_l) is added if both are on the same site.

b. The edge (T_k, T_l, n) is added in both the sites, if T_k and T_l are on different sites.

Therefore, if the algorithm were able to collect all the local wait-for graphs at the same instant, it would definitely discover a cycle in the constructed graph, in case there is a circular wait at that instant. If there is a circular wait at the instant when the algorithm began execution, none of the edges participating in that cycle can disappear until the algorithm finishes. Therefore, even though the algorithm cannot collect all the local graphs at the same instant, any cycle which existed just before it started will anyway be detected.

19.10 Consider a relation that is fragmented horizontally by *plant_number*:

employee (*name, address, salary, plant_number*)

Assume that each fragment has two replicas: one stored at the New York site and one stored locally at the plant site. Describe a good processing strategy for the following queries entered at the San Jose site.

a. Find all employees at the Boca plant.

b. Find the average salary of all employees.

c. Find the highest-paid employee at each of the following sites: Toronto, Edmonton, Vancouver, Montreal.

d. Find the lowest-paid employee in the company.

Answer:

a. i. Send the query $\Pi_{name}(employee)$ to the Boca plant.
   ii. Have the Boca location send back the answer.

b. i. Compute average at New York.
   ii. Send answer to San Jose.

c. i. Send the query to find the highest salaried employee to Toronto, Edmonton, Vancouver, and Montreal.
   ii. Compute the queries at those sites.
   iii. Return answers to San Jose.

d. i. Send the query to find the lowest salaried employee to New York.
   ii. Compute the query at New York.
   iii. Send answer to San Jose.

19.11 Compute $r \ltimes s$ for the relations of Figure 19.9.
Answer: The result is as follows.

$r \ltimes s =$

A	B	C
1	2	3
5	3	2

19.12 Give an example of an application ideally suited for the cloud and another that would be hard to implement successfully in the cloud. Explain your answer.
Answer: Any application that is easy to partition, and does not need strong guarantees of consistency across partitions, is ideally suited to the cloud. For example, Web-based document storage systems (like Google docs), and Web based email systems (like Hotmail, Yahoo! mail or GMail), are ideally suited to the cloud. The cloud is also ideally suited to certain kinds of data analysis tasks where the data is already on the cloud; for example, the Google Map-Reduce framework, and Yahoo! Hadoop are widely used for data analysis of Web logs such as logs of URLs clicked by users.
Any database application that needs transactional consistency would be hard to implement successfully in the cloud; examples include bank records, academic records of students, and many other types of organizational records.

19.13 Given that the LDAP functionality can be implemented on top of a database system, what is the need for the LDAP standard?
Answer: The reasons are:

a. Directory access protocols are simplified protocols that cater to a limited type of access to data.

b. Directory systems provide a simple mechanism to name objects in a hierarchical fashion which can be used in a distributed directory system to specify what information is stored in each of the directory servers. The directory system can be set up to automatically forward queries made at one site to the other site, without user intervention.

19.14 Consider a multidatabase system in which it is guaranteed that at most one global transaction is active at any time, and every local site ensures local serializability.

a. Suggest ways in which the multidatabase system can ensure that there is at most one active global transaction at any time.

b. Show by example that it is possible for a nonserializable global schedule to result despite the assumptions.

Answer:

a. We can have a special data item at some site on which a lock will have to be obtained before starting a global transaction. The lock should be released after the transaction completes. This ensures the single active global transaction requirement. To reduce dependency on that particular site being up, we can generalize the solution by having an election scheme to choose one of the currently up sites to be the co-ordinator, and requiring that the lock be requested on the data item which resides on the currently elected co-ordinator.

b. The following schedule involves two sites and four transactions. T_1 and T_2 are local transactions, running at site 1 and site 2 respectively. T_{G1} and T_{G2} are global transactions running at both sites. X_1, Y_1 are data items at site 1, and X_2, Y_2 are at site 2.

T_1	T_2	T_{G1}	T_{G2}
write(Y_1)			
		read(Y_1)	
		write(X_2)	
	read(X_2)		
	write(Y_2)		
			read(Y_2)
			write(X_1)
read(X_1)			

In this schedule, T_{G2} starts only after T_{G1} finishes. Within each site, there is local serializability. In site 1, $T_{G2} \rightarrow T_1 \rightarrow T_{G1}$ is a serializability order. In site 2, $T_{G1} \rightarrow T_2 \rightarrow T_{G2}$ is a serializability order. Yet the global schedule schedule is non-serializable.

19.15 Consider a multidatabase system in which every local site ensures local serializability, and all global transactions are read only.

a. Show by example that nonserializable executions may result in such a system.

b. Show how you could use a ticket scheme to ensure global serializability.

Answer:

a. The same system as in the answer to Exercise 19.14 is assumed, except that now both the global transactions are read-only. Consider the schedule given below.

T_1	T_2	T_{G1}	T_{G2}
			read(X_1)
write(X_1)			
		read(X_1)	
		read(X_2)	
	write(X_2)		
			read(X_2)

Though there is local serializability in both sites, the global schedule is not serializable.

b. Since local serializability is guaranteed, any cycle in the system wide precedence graph must involve at least two different sites, and two different global transactions. The ticket scheme ensures that whenever two global transactions access data at a site, they conflict on a data item (the ticket) at that site. The global transaction manager controls ticket access in such a manner that the global transactions execute with the same serializability order in all the sites. Thus the chance of their participating in a cycle in the system wide precedence graph is eliminated.

21

PARALLEL AND DISTRIBUTED DATABASES

Exercise 21.1 Give brief answers to the following questions:

1. What are the similarities and differences between parallel and distributed database management systems?
2. Would you expect to see a parallel database built using a wide-area network? Would you expect to see a distributed database built using a wide-area network? Explain.
3. Define the terms *scale-up* and *speed-up*.
4. Why is a shared-nothing architecture attractive for parallel database systems?
5. The idea of building specialized hardware to run parallel database applications received considerable attention but has fallen out of favor. Comment on this trend.
6. What are the advantages of a distributed database management system over a centralized DBMS?
7. Briefly describe and compare the Client-Server and Collaborating Servers architectures.
8. In the Collaborating Servers architecture, when a transaction is submitted to the DBMS, briefly describe how its activities at various sites are coordinated. In particular, describe the role of transaction managers at the different sites, the concept of *subtransactions*, and the concept of *distributed transaction atomicity*.

Answer 21.1 1. Parallel and distributed database management systems are similar in form, yet differ in function. The form of both types of DBMS must include direct access to both multiple storage devices and multiple processors. Note the stringency of the direct access provision. Not only does the physical architecture need to include multiple storage units and processors, but the operating system must allow the the parallel or distributed DBMS to store or process data using a particular disk or processor. An operating system that internally manages multiplicity and presents a single disk, single processor platform could not support a parallel or distributed DBMS.

Both types of DBMS directly manipulate multiple storage devices and both have the capacity to perform operations in a non-sequential fashion, but each does so for a different functional purpose. A DBMS is made parallel primarily to improve performance by allowing non-interdependent operations to execute simultaneously. The data locations are chosen to optimize input/output requests from the processors. A DBMS is made

distributed primarily to store the data in a particular location which then determines the choice of processor. Multiple locations serve as a safety net should one site fail, and provide quicker access to local data for geographically large organizations.

2. No, it would be unlikely to find a parallel database system built using a wide-area network. Any performance gains from executing operations simultaneously would surely be lost by excessive transportation costs.

   Yes, a distributed database is likely to be built using a wide-area network. Multiple copies of the data may be stored at geographically distant locations to optimize local data requests and enhance availability in the event one site fails.

3. *Speed-up* is defined as the proportional decrease in processing time due to an increase in number of disks or processors, with the amount of data held constant. In other words, for a fixed amount of data, speed-up measures how much the speed increases due to additional processors or disks.

   Scale-up is defined as the proportional increase in data processing ability due to an increase in the number of disks or processors, with the processing time held constant. In other words, in a fixed amount of time, scale-up measures the data capacity increase due to additional processors or disks.

4. The shared-nothing architecture is attractive because it allows for linear *scale-up* and *speed-up* for an arbitrary number of processors. In contrast, the shared-memory architecture can only provide these performance gains for a fixed number of processors. After a certain point, memory contention degrades performance and the gains from the shared memory approach are significantly less than those from shared-nothing alternative.

5. The production of specialized hardware requires a large capital investment which in turn requires a large market for success. While the market for database products is huge, the sub-segment of customers willing to pay a premium for parallel performance gains is smaller. Moreover, this sub-segment already has access to high-performance at a lower cost with stock hardware. Recall that using standard CPUs and interconnects in a shared nothing architecture allows for linear *speed-up* and *scale-up*. Since specialized hardware cannot provide better performance per cost, nor can it keep pace with the rapid development of stock hardware, there is a subsequent lack of development interest.

6. A distributed database system is superior to its centralized counterpart for several reasons. First, data may be replicated at multiple locations which provides increased reliability in the event that one site fails. Second, if the organization served by the DBMS is geographically diverse and access patterns are localized, storing the data locally will greatly reduce transportation costs thus improving performance. Finally, distributing data in a large organization allows for greater local autonomy so that issues of only local concern may be handled locally. A centralized database would need to coordinate too many details, e.g. ensuring no conflicts everytime someone chooses a name for a database object.

7. The client-server architecture draws a sharp distinction between the user and the data storage. The client side contains a front end for the purpose of generating queries to be sent to the server, which processes the query and responds with the data. A collaborating-servers architecture differs in that there is a collection of servers capable of processing queries. In addition, if data is needed from multiple servers, each unit is capable of decomposing a large query into smaller queries, and sending them to the ap-

propriate location. Thus in the collaborating-server architecture, servers not only store data and process queries, but they may also act as users of other servers.

8. A transaction submitted to a collaborating-server architecture is first evaluated to determine where the data is located and what optimized sub-queries will retrieve it most efficiently. The primary server, the recipient of the initial query, then begins a transaction and starts acquiring the necessary locks. The primary transaction manager acquires local locks in the normal fashion and issues *subtransaction* requests to the remote servers. The remote servers set about acquiring the necessary locks for the locally optimized plan and communicate back to the primary transaction manager.

   Once the primary transaction manager hears positive results from every remote transaction manager: the recovery data is stored in a safe place, the transaction commits, and executes it its entirety. If at least one remote transaction manager replies with an abort message or fails to respond at all, the primary transaction manager aborts the entire transaction. *Distributed transaction atomicity* is then guaranteed in that either the entire transaction executes everywhere or none of it executes anywhere.

Exercise 21.2 Give brief answers to the following questions:

1. Define the terms *fragmentation* and *replication*, in terms of where data is stored.
2. What is the difference between *synchronous* and *asynchronous* replication?
3. Define the term *distributed data independence*. Specifically, what does this mean with respect to querying and with respect to updating data in the presence of data fragmentation and replication?
4. Consider the *voting* and *read-one write-all* techniques for implementing synchronous replication. What are their respective pros and cons?
5. Give an overview of how asynchronous replication can be implemented. In particular, explain the terms *capture* and *apply*.
6. What is the difference between log-based and procedural approaches to implementing capture?
7. Why is giving database objects unique names more complicated in a distributed DBMS?
8. Describe a catalog organization that permits any replica (of an entire relation or a fragment) to be given a unique name and that provides the naming infrastructure required for ensuring distributed data independence.
9. If information from remote catalogs is cached at other sites, what happens if the cached information becomes outdated? How can this condition be detected and resolved?

Answer 21.2 Answer omitted.

Exercise 21.3 Consider a parallel DBMS in which each relation is stored by horizontally partitioning its tuples across all disks.

Employees(*eid:* `integer`, *did:* `integer`, *sal:* `real`)
Departments(*did:* `integer`, *mgrid:* `integer`, *budget:* `integer`)

The *mgrid* field of Departments is the *eid* of the manager. Each relation contains 20-byte tuples, and the *sal* and *budget* fields both contain uniformly distributed values in the range 0 to 1,000,000. The Employees relation contains 100,000 pages, the Departments relation contains 5,000 pages, and each processor has 100 buffer pages of 4,000 bytes each. The cost of one page I/O is t_d, and the cost of shipping one page is t_s; tuples are shipped in units of one page by waiting for a page to be filled before sending a message from processor i to processor j. There are no indexes, and all joins that are local to a processor are carried out using a sort-merge join. Assume that the relations are initially partitioned using a round-robin algorithm and that there are 10 processors.

For each of the following queries, describe the evaluation plan briefly and give its cost in terms of t_d and t_s. You should compute the total cost across all sites as well as the 'elapsed time' cost (i.e., if several operations are carried out concurrently, the time taken is the maximum over these operations).

1. Find the highest paid employee.
2. Find the highest paid employee in the department with *did* 55.
3. Find the highest paid employee over all departments with *budget* less than 100,000.
4. Find the highest paid employee over all departments with *budget* less than 300,000.
5. Find the average salary over all departments with *budget* less than 300,000.
6. Find the salaries of all managers.
7. Find the salaries of all managers who manage a department with a budget less than 300,000 and earn more than 100,000.
8. Print the *eid*s of all employees, ordered by increasing salaries. Each processor is connected to a separate printer, and the answer can appear as several sorted lists, each printed by a different processor, as long as we can obtain a fully sorted list by concatenating the printed lists (in some order).

Answer 21.3 The round-robin partitioning implies that every tuple has a equal probability of residing at each processor. Moreover, since the *sal* field of Employees and *budget* field of Departments are uniformly distributed on 0 to 1,000,000, each processor must also have a uniform distribution on this range. Also note that processing a partial page incurs the same cost as processing an entire page and the cost of writing out the result is uniformly ignored. Finally, recall that elapsed time is the maximum time taken for any one processor to complete its task.

1. Find the highest paid employee.

   Plan: Conduct a complete linear scan of the Employees relation at each processor retaining only the tuple with the highest value in *sal* field. All processors except one then send their result to a chosen processor which selects the tuple with the highest value of *sal*.

$$\begin{aligned} Total\ Cost &= (\#\ CPUs) * (Emp\ pg\ /CPU) * (I/O\ cost) \\ &+ (\#\ CPUs - 1) * (send\ cost) \end{aligned}$$

$$\begin{aligned} &= (10 * 10,000 * t_d) + (9 * t_s) \\ &= 100,000 * t_d + 9 * t_s \end{aligned}$$

$$Elapsed\ Time = 10,000 * t_d + t_s$$

2. Find the highest paid employee in the department with *did* 55.

   Plan: Conduct a complete linear scan of the Employees relation at each processor retaining only the tuple with the highest value in *sal* field and a *did* field equal to 55. All processors except one then send their result to a chosen processor which selects the tuple with the highest value of *sal*.

   Total Cost: The answer is the same as for part 1 above. Even if no qualifying tuples are found at a given processor, a page should still be sent from nine processors to a chosen tenth. The page will either contain a real tuple or if a processor fails to find any tuple with *did* equal to 55, a generated tuple with *sal* equal to -1 will suffice. Note that the chosen processor must also account for the case where no tuple qualifies, simply by ignoring any tuple with *sal* equal to -1 in its final selection.

   Elapsed Time: The elapsed time is also the same as for part 1 above.

3. Find the highest paid employee over all departments with *budget* less than 100,000.

   Plan: First, conduct a complete linear scan of the Departments relation at each processor retaining only the *did* fields from tuples with *budget* less than 100,000. Recall that Departments is uniformly distributed on the *budget* field from 0 to 1,000,000, thus each processor will retain only 10% of its 500 Departments pages. Since the *did* field is 1/3 of a Departments tuple, the scan will result in approximately 16.7 pages which rounds up to 17.

   Second, each processor sends its 17 pages of retained *did* field tuples to every other processor which subsequently stores them. 10 processors send 17 pages to 9 other processors for a total of 1,530 sends. After sending, each processor has 170 (partially filled) pages of Departments tuples.

   Third, each processor joins the *did* field tuples with the Employees relation retaining only the joined tuple with the highest value in the *sal* field. Let $M = 170$ represent the number of Departments pages and $N = 10,000$ represent the number of Employees pages at each processor. Since the number of buffer pages, $100 \geq \sqrt{N}$, the refined Sort-Merge may be used for a join cost of 30,510 at each processor.

   Fourth, all processors except one then send their result to a chosen processor which selects the tuple with the highest value of *sal*.

$$\begin{aligned} Total\ Cost &= scan\ Dept\ for\ tuples\ with\ budget < 100,000 \\ &+ sending\ did\ field\ tuples\ from\ 10\ processors\ to\ 9\ others \\ &+ storing\ did\ field\ tuples\ at\ each\ processor \\ &+ joining\ with\ Emp\ and\ selecting\ max(sal)\ tuple \\ &+ sending\ local\ results\ to\ the\ chosen\ processor \\ \\ &= (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \end{aligned}$$

$$
\begin{aligned}
& & & 10 * 500 * t_d \\
& & & 5,000 * t_d \\
& & + & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ did\ pgs) * t_s \\
& & & 10 * 9 * 17 * t_s \\
& & & 1,530 * t_s \\
& & + & (\#\ CPU\ storing) * (170\ did\ pgs) * (I/O\ cost) \\
& & & 10 * 170 * t_d \\
& & & 1,700 * t_d \\
& & + & (\#\ CPU\ joining) * (join\ cost) \\
& & & 10 * (3 * (170 + 10,000) * t_d) \\
& & & 10 * 30,510 * t_d \\
& & & 305,100 * t_d \\
& & + & (\#\ CPUs - 1) * (send\ cost) \\
& & & 9 * t_s \\
& & = & 5,000 * t_d + 1,530 * t_s + 1,700 * t_d + 305,100 * t_d + 9 * t_s \\
& & = & 311,800 * t_d + 1,539 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time & = & 500 * t_d + 153 * t_s + 170 * t_d + 30,510 * t_d + t_s \\
& = & 31,180 * t_d + 154 * t_s
\end{aligned}
$$

4. Find the highest paid employee over all departments with *budget* less than 300,000.

   Plan: The evaluation of this query is identical to that in part 3 except that the probability of a Departments tuple's *budget* field being selected in step one is multiplied by three. There are then 50 pages retained by each processor and sent to every other processor for joins and maximum selection.

$$
\begin{aligned}
Total\ Cost & = & scan\ Dept\ for\ tuples\ with\ budget < 300,000 \\
& + & sending\ did\ field\ tuples\ from\ 10\ processors\ to\ 9\ others \\
& + & storing\ did\ field\ tuples\ at\ each\ processor \\
& + & joining\ with\ Emp\ and\ selecting\ max(sal)\ tuple \\
& + & sending\ local\ results\ to\ the\ chosen\ processor \\
& = & (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& & 10 * 500 * t_d \\
& & 5,000 * t_d \\
& + & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (50\ did\ pgs) * t_s \\
& & 10 * 9 * 50 * t_s \\
& & 4,500 * t_s \\
& + & (\#\ CPU\ storing) * (500\ did\ pgs) * (I/O\ cost) \\
& & 10 * 500 * t_d
\end{aligned}
$$

$$
\begin{aligned}
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 10 * (3 * (500 + 10,000) * t_d) \\
& 10 * 31,500 * t_d \\
& 315,000 * t_d \\
+ \quad & (\#\ CPUs\ - 1) * (send\ cost) \\
& 9 * t_s \\
= \quad & 5,000 * t_d + 4,500 * t_s + 5,000 * t_d + 315,000 * t_d + 9 * t_s \\
= \quad & 325,000 * t_d + 4,509 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time \quad = \quad & 500 * t_d + 450 * t_s + 500 * t_d + 31,500 * t_d + t_s \\
= \quad & 32,500 * t_d + 451 * t_s
\end{aligned}
$$

5. Find the average salary over all departments with *budget* less than 300,000.

   Plan: The first two steps in evaluating this query are identical to part 4. Steps three and four differ in that the desired result is an average instead of a maximum.

   First, each processor conducts a complete linear scan of the Departments relation retaining only the *did* field from tuples with a *budget* field less than 300,000. Second, each processor sends its result pages to every other processor. Third, each processor joins the *did* field tuples with the Employees relation and retains a running sum and count of the *sal* field. Fourth, each processor except one sends its sum and count to a chosen processor which divides the total sum by the total count to obtain the average. The cost is identical to part 4 above.

6. Find the salaries of all managers.

   Plan: First, conduct a complete linear scan of the Departments relation at each processor retaining only the *mgrid* field for all tuples. Since the *mgrid* field is 1/3 of each tuple, there will be 167 (rounded up) resulting pages. Second, each processor sends its result pages to every other processor which subsequently stores them. Third, each processor joins the *mgrid* field tuples with Employees thus obtaining the salaries of all managers.

$$
\begin{aligned}
Total\ Cost \quad = \quad & scan\ Dept\ for\ mgrid\ fields \\
+ \quad & sending\ mgrid\ field\ tuples\ from\ 10\ processors\ to\ 9\ others \\
+ \quad & storing\ mgrid\ field\ tuples\ at\ each\ processor \\
+ \quad & joining\ with\ Emp \\
= \quad & (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& 10 * 500 * t_d \\
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (167\ mgrid\ pgs) * t_s \\
& 10 * 9 * 167 * t_s \\
& 15,030 * t_s \\
+ \quad & (\#\ CPU\ storing) * (1,670\ mgrid\ pgs) * (I/O\ cost)
\end{aligned}
$$

$$
\begin{aligned}
& 10 * 1,670 * t_d \\
& 16,700 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 10 * (3 * (1,670 + 10,000) * t_d) \\
& 10 * 35,010 * t_d \\
& 350,100 * t_d \\
= \quad & 5,000 * t_d + 15,030 * t_s + 16,700 * t_d + 350,100 * t_d \\
= \quad & 386,830 * t_d + 15,030 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time \quad = \quad & 500 * t_d + 1,503 * t_s + 1,670 * t_d + 35,010 * t_d \\
= \quad & 38,683 * t_d + 1,503 * t_s
\end{aligned}
$$

7. Find the salaries of all managers who manage a department with a budget less than 300,000 and earn more than 100,000.

   Plan: The evaluation of this query is similar to that of part 6. The additional selection condition on the *budget* field is applied in step one and serves to reduce the number of pages sent and joined in steps two and three. The additional selection condition on the *sal* field is applied during the join in step three and has no effect on the final cost.

   First, conduct a complete linear scan of the Departments relation at each processor retaining only the *mgrid* field for all tuples. Since the *mgrid* field is 1/3 of each tuple and there are 150 qualifying Departments pages at each processor, there will be 50 resulting pages. Second, each processor sends its result pages to every other processor which subsequently stores them. Third, each processor joins the *mgrid* field tuples with Employees thus obtaining the salaries of all managers.

$$
\begin{aligned}
Total\ Cost \quad = \quad & scan\ Dept\ for\ mgrid\ fields \\
+ \quad & sending\ mgrid\ field\ tuples\ from\ 10\ processors\ to\ 9\ others \\
+ \quad & storing\ mgrid\ field\ tuples\ at\ each\ processor \\
+ \quad & joining\ with\ Emp \\
= \quad & (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& 10 * 500 * t_d \\
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (50\ mgrid\ pgs) * t_s \\
& 10 * 9 * 50 * t_s \\
& 4,500 * t_s \\
+ \quad & (\#\ CPU\ storing) * (500\ mgrid\ pgs) * (I/O\ cost) \\
& 10 * 500 * t_d \\
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 10 * (3 * (500 + 10,000) * t_d)
\end{aligned}
$$

$$
\begin{aligned}
& 10 * 31,500 * t_d \\
& 315,000 * t_d \\
= \; & 5,000 * t_d + 4,500 * t_s + 5,000 * t_d + 315,000 * t_d \\
= \; & 325,000 * t_s + 4,500 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 500 * t_d + 450 * t_s + 500 * t_d + 31,500 * t_d \\
&= 32,500 * t_d + 450 * t_s
\end{aligned}
$$

8. Print the *eids* of all employees, ordered by increasing salaries. Each processor is connected to a separate printer, and it is acceptable to have the answer in the form of several sorted lists, each printed by a different processor, as long as we can obtain a fully sorted list by concatenating the printed lists (in some order).

   Plan: At each processor, sort the Employees relation by the *sal* field and print the result. Note that the refined Sort-Merge join may be applied without the on-the-fly merge to sort at a cost of $3 * M * t_d$.

$$
\begin{aligned}
Total\ Cost &= (\#\ CPU\ sorting) * (sort\ cost) \\
&= 10 * (3 * 10,000 * t_d) \\
&= 300,000 * t_d
\end{aligned}
$$

$$
Elapsed\ Time = 30,000 * t_d
$$

Exercise 21.4 Consider the same scenario as in Exercise 21.3, except that the relations are originally partitioned using range partitioning on the *sal* and *budget* fields.

Answer 21.4 Answer omitted.

Exercise 21.5 Repeat Exercises 21.3 and 21.4 with the number of processors equal to (i) 1 and (ii) 100.

Answer 21.5 Repeat of Exercise 21.3

Recall that the round-robin distribution algorithm implies that the tuples are uniformly distributed across processors. Moreover, since the Employees and Departments relations *sal* and *budget* fields are uniformly distributed on 0 to 1,000,000, each processor must also have a uniform distribution on this range. Since elapsed time figures are redundant for the one processor case they are omitted. Also, assume for simplicity that the single processor has enough buffer pages for the Sort-Merge join algorithm, i.e., 317. Finally, for the 100 processor case, the plans are nearly identical to Exercise 20.3 and thus are also omitted.

(i) Assuming there is only 1 processor

(ii) Assuming there are 100 processors

1. Find the highest paid employee

   (i) Plan: Conduct a complete linear scan of all Employees tuples retaining only the one with the highest *sal* value.

$$Cost = (\#\ Emp\ pgs) * (I/O\ cost) = 100,000 * t_d$$

   (ii)

$$\begin{aligned} Total\ Cost &= (\#\ CPUs) * (Emp\ pgs/CPU) * (I/O\ cost) \\ &+ (\#\ CPUs - 1) * (send\ cost) \\ &= (100 * 1,000 * t_d) + (99 * t_s) \\ &= 100,000 * t_d + 99 * t_s \end{aligned}$$

$$Elapsed\ Time = 1,000 * t_d + t_s$$

2. Find the highest paid employee in the department with *did* 55.

   (i) Plan: Conduct a complete linear scan of all Employees tuples retaining only the one with the highest *sal* value and *did* field equal to 55.

$$\begin{aligned} Cost &= (\#\ Emp\ pgs) * (I/O\ cost) \\ &= 100,000 * t_d \end{aligned}$$

   (ii) Total and elapsed costs are identical to part 1 above.

3. Find the highest paid employee over all departments with *budget* less than 100,000.

   (i) Plan: join the Employees and Departments relations retaining only the one with the highest salary and budget less than 100,000.

$$\begin{aligned} Cost &= 3 * (\#\ Dept\ pgs\ +\#\ Emp\ pgs) * (I/O\ cost) \\ &= 3 * (100,000 + 5,000) * t_d \\ &= 315,000 * t_d \end{aligned}$$

   (ii)

$$\begin{aligned} Total\ Cost &= scan\ Dept\ for\ tuples\ with\ budget < 100,000 \\ &+ sending\ did\ pgs\ from\ 100\ processors\ to\ 99\ others \\ &+ storing\ did\ pgs\ at\ each\ processor \\ &+ joining\ with\ Emp\ and\ selecting\ max(sal)\ tuple \\ &+ sending\ local\ results\ to\ the\ chosen\ processor \\ &= (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\ &\quad 100 * 50 * t_d \\ &\quad 5,000 * t_d \end{aligned}$$

$$
\begin{aligned}
&+ \quad (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (2\ did\ pgs) * t_s \\
&\quad 100 * 99 * 2 * t_s \\
&\quad 19,800 * t_s \\
&+ \quad (\#\ CPU\ storing) * (200\ did\ pgs) * (I/O\ cost) \\
&\quad 100 * 200 * t_d \\
&\quad 20,000 * t_d \\
&+ \quad (\#\ CPU\ joining) * (join\ cost) \\
&\quad 100 * (3 * (200 + 1,000) * t_d) \\
&\quad 100 * 3,600 * t_d \\
&\quad 360,000 * t_d \\
&+ \quad (\#\ CPUs - 1) * (send\ cost) \\
&\quad 99 * t_s \\
&= \quad 5,000 * t_d + 19,800 * t_s + 20,000 * t_d + 360,000 * t_d + 99 * t_s \\
&= \quad 385,000 * t_d + 19,899 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 198 * t_s + 200 * t_d + 3,600 * t_d + t_s \\
&= 3,850 * t_d + 199 * t_s
\end{aligned}
$$

4. Find the highest paid employee over all departments with a budget less than 300,000.
(i) Plan: join the Employees and Departments relations retaining only the one with the highest salary and budget less than 300,000.

$$
\begin{aligned}
Cost &= 3 * (\#\ Dept\ pgs\ +\#\ Emp\ pgs) * (I/O\ cost) \\
&= 3 * (100,000 + 5,000) * t_d \\
&= 315,000 * t_d
\end{aligned}
$$

(ii)

$$
\begin{aligned}
Total\ Cost &= scan\ Dept\ for\ tuples\ with\ budget < 300,000 \\
&+ sending\ did\ field\ pgs\ from\ 100\ processors\ to\ 99\ others \\
&+ storing\ did\ field\ pgs\ at\ each\ processor \\
&+ joining\ with\ Emp\ and\ selecting\ max(sal)\ tuple \\
&+ sending\ local\ results\ to\ the\ chosen\ processor \\
&= (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
&\quad 100 * 50 * t_d \\
&\quad 5,000 * t_d \\
&+ \quad (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (5\ did\ pgs) * t_s \\
&\quad 100 * 99 * 5 * t_s \\
&\quad 49,500 * t_s \\
&+ \quad (\#\ CPU\ storing) * (500\ did\ pgs) * (I/O\ cost)
\end{aligned}
$$

$$
\begin{aligned}
& 100 * 500 * t_d \\
& 50,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 100 * (3 * (500 + 1,000) * t_d) \\
& 10 * 4,500 * t_d \\
& 450,000 * t_d \\
+ \quad & (\#\ CPUs - 1) * (send\ cost) \\
& 99 * t_s \\
= \quad & 5,000 * t_d + 49,500 * t_s + 50,000 * t_d + 450,000 * t_d + 99 * t_s \\
= \quad & 495,000 * t_d + 49,599 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time \quad = \quad & 50 * t_d + 495 * t_s + 500 * t_d + 4,500 * t_d + t_s \\
= \quad & 4,950 * t_d + 496 * t_s
\end{aligned}
$$

5. Find the average salary over all departments with budget less than 300,000.

   (i) Plan: join the Employees and Departments relations retaining a running sum and count of the *sal* field for join tuples with a *budget* field less than 300,000. Divide the sum by the count to obtain the average.

$$
\begin{aligned}
Cost \quad = \quad & 3 * (\#\ Dept\ pgs + \#\ Emp\ pgs) * (I/O\ cost) \\
= \quad & 3 * (100,000 + 5,000) * t_d \\
= \quad & 315,000 * t_d
\end{aligned}
$$

   (ii) The cost is identical to part 4 above.

6. Find the salaries of all managers.

   (i) Plan: join the Employees and Departments relations at each processor retaining only those join tuples with *eid* equal to *mgrid*.

$$
\begin{aligned}
Cost \quad = \quad & 3 * (\#\ Dept\ pgs + \#\ Emp\ pgs) * (I/O\ cost) \\
= \quad & 3 * (100,000 + 5,000) * t_d \\
= \quad & 315,000 * t_d
\end{aligned}
$$

   (ii)

$$
\begin{aligned}
Total\ Cost \quad = \quad & scan\ Dept\ for\ mgrid\ fields \\
+ \quad & sending\ mgrid\ pgs\ from\ 100\ processors\ to\ 99\ others \\
+ \quad & storing\ mgrid\ pgs\ at\ each\ processor \\
+ \quad & joining\ with\ Emp \\
= \quad & (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& 100 * 50 * t_d
\end{aligned}
$$

$$
\begin{aligned}
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ mgrid\ pgs) * t_s \\
& 100 * 99 * 17 * t_s \\
& 168,300 * t_s \\
+ \quad & (\#\ CPU\ storing) * (170\ mgrid\ pgs) * (I/O\ cost) \\
& 100 * 170 * t_d \\
& 17,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 100 * (3 * (1,700 + 1,000)t_d) \\
& 100 * 8,100 * t_d \\
& 810,000 * t_d \\
= \quad & 5,000 * t_d + 168,300 * t_s + 17,000 * t_d + 810,000 * t_d \\
= \quad & 832,000 * t_d + 168,300 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time \quad = \quad & 50 * t_d + 1,683 * t_s + 170 * t_d + 8,100 * t_d \\
= \quad & 8,320 * t_d + 1,683 * t_s
\end{aligned}
$$

7. Find the salaries of all managers who manage a department with a budget less than 300,000 and earn more than 100,000.

   (i) Plan: join the Employees and Department relations retaining only those joined tuples with $budget < 300{,}000$ and $sal > 100{,}000$.

$$
\begin{aligned}
Cost \quad = \quad & 3 * (\#\ Dept\ pages + \#\ Emp\ pages) * (I/O\ cost) \\
= \quad & 3 * (100,000 + 5,000) * t_d \\
= \quad & 315,000 * t_d
\end{aligned}
$$

   (ii)

$$
\begin{aligned}
Total\ Cost \quad = \quad & scan\ Dept\ for\ mgrid\ fields \\
+ \quad & sending\ mgrid\ pgs\ from\ 100\ processors\ to\ 99\ others \\
+ \quad & storing\ mgrid\ pgs\ at\ each\ processor \\
+ \quad & joining\ with\ Emp \\
= \quad & (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& 100 * 50 * t_d \\
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (5\ mgrid\ pgs) * t_s \\
& 100 * 99 * 5 * t_s \\
& 49,500 * t_s \\
+ \quad & (\#\ CPU\ storing) * (500\ mgrid\ pgs) * (I/O\ cost) \\
& 100 * 500 * t_d
\end{aligned}
$$

$$
\begin{aligned}
& 50,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 100 * (3 * (500 + 1,000)t_d) \\
& 100 * 4,500 * t_d \\
& 450,000 * t_d \\
= \quad & 5,000 * t_d + 49,500 * t_s + 50,000 * t_d + 450,000 * t_d \\
= \quad & 505,000 * t_s + 49,500 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 495 * t_s + 500 * t_d + 4,500 * t_d \\
&= 5,050 * t_d + 495 * t_s
\end{aligned}
$$

8. Print the eids of all employees, ordered by increasing salaries.
   (i) Plan: sort the Employees relation using salary as a key and print the result.

$$
\begin{aligned}
Cost &= (100,000) * (sortcost) \\
&= 300,000 * t_d
\end{aligned}
$$

   (ii)

$$
\begin{aligned}
Total\ Cost &= (\#\ CPU\ sorting) * (sort\ cost) \\
&= 100 * (3 * 1,000 * t_d) \\
&= 300,000 * t_d
\end{aligned}
$$

$$
Elapsed\ Time = 3,000 * t_d
$$

Repeat of Exercise 21.4:

Recall that in Exercise 21.3 the range partitioning places tuples with either a *sal* or *budget* field between 0 and 10,000 at processor 1, between 10,001 and 20,000 at the processor 2, etc. The uniform distribution of values in *sal* and *budget* implies that there are equal numbers of tuples at each processor. Assuming 100 processors, there are 1,000 Employee tuples and 50 department tuples at each processor. Assuming there is only one processor implies partitioning is meaningless, thus the answers for part (i) are identical to those from part(i) directly above and are omitted.

The answers below assume that there are 100 processors.

1. Find the highest paid employee.
   Plan: The tuple with the highest *sal* value is located at processor 100. Conduct a complete linear scan of the Employees relation there retaining the tuple with the highest value in the *sal* field.

$$\begin{aligned} Total\ Cost &= (\#\ of\ Emp\ pgs\ at\ CPU\ 100) * (I/O\ cost) \\ &= 1,000 * t_d \end{aligned}$$

$$Elapsed\ Time = 1,000 * t_d$$

2. Find the highest paid employee in the department with *did* 55.

   Plan: Since there is no guarantee that such a tuple might exist at any given processor, conduct a complete linear scan of all Employees tuples at each processor retaining the one with the highest *sal* value and *did* 55. Each processor except one should then send their result to a chosen processor which selects the tuple with the highest value in the *sal* field.

$$\begin{aligned} Total\ Cost &= (\#\ CPU\ scanning) * (\#\ of\ Emp\ pgs/CPU) * (I/O\ cost) \\ &+ (\#CPUs - 1) * (sendcost) \\ &= 100 * 1,000 * t_d \\ &+ 99 * t_s \\ &= 100,000 * t_d + 99 * t_s \end{aligned}$$

$$Elapsed\ Time = 1,000 * t_d + t_s$$

3. Find the highest paid employee over all departments with *budget* less than 100,000.

   Plan: Department tuples with a *budget* field less than 100,000 must be located at processors 1 through 10. The highest paid employees are located at the higher numbered processors, however; as in the 2. above, there is no guarantee that any processor has an Employees tuple with a particular *did* field value. So, processors 1 through 10 must conduct a complete linear scan of Departments retaining only the *did* field. The results are then sent to all processors which store and join them with the Employees relation retaining only the join tuple with the highest *sal* value. Finally, each processor except one sends the result to a chosen processor which selects the Employees tuple with the highest *sal* value.

$$\begin{aligned} Total\ Cost &= scan\ Dept\ for\ did\ fields\ at\ first\ ten\ CPUs \\ &+ sending\ did\ pgs\ from\ 10\ CPUs\ to\ 99\ CPUs \\ &+ storing\ did\ pgs\ at\ each\ processor \\ &+ joining\ did\ with\ Emp \\ &+ sending\ local\ results\ to\ chosen\ processor \\ &= (\#\ CPUs\ w/budget < 100,000) * (\#\ Dept\ pgs) * (I/Ocost) \\ &\quad (10\ CPUs) * (50\ pgs/CPU) * t_d \\ &\quad 10 * 50 * t_d \\ &\quad 500 * t_d \end{aligned}$$

$$
\begin{aligned}
&+ \quad (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ did\ pgs) * t_s \\
&\quad 10 * 99 * 17 * t_s \\
&\quad 10 * 1,683 * t_s \\
&\quad 16,830 * t_s \\
&+ \quad (\#\ CPU\ storing) * (170\ did\ pgs) * (I/O\ cost) \\
&\quad 100 * 170 * t_d \\
&\quad 17,000 * t_d \\
&+ \quad (\#\ CPU\ joining) * (join\ cost) \\
&\quad 100 * (3 * (170 * 1,000) * t_d) \\
&\quad 351,000 * t_d \\
&+ \quad (\#\ CPUs\ -1) * (send\ cost) \\
&\quad 99 * t_s \\
&= \quad 500 * t_d + 16,830 * t_s + 17,000 * t_d + 351,000 * t_d + 99 * t_s \\
&= \quad 368,500 * t_d + 16,929 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 1,683 * t_s + 170 * t_d + 3,510 * t_d + t_s \\
&= 3,730 * t_d + 1,684 * t_s
\end{aligned}
$$

4. Find the highest paid employee over all departments with a budget less than 300,000.
   Plan: The plan is identical to that for part 3 above except that now the first 30 processors must create relations of *did* fields and send them to all other processors.

$$
\begin{aligned}
Total\ Cost &= scan\ Dept\ for\ did\ fields\ at\ first\ thirty\ CPUs \\
&+ sending\ did\ field\ pgs\ from\ 30\ CPUs\ to\ 99\ CPUs \\
&+ storing\ did\ field\ pgs\ at\ each\ processor \\
&+ joining\ did\ field\ tuples\ with\ Emp\ tuples \\
&+ sending\ local\ results\ to\ chosen\ processor \\
&= (\#\ CPUs\ w/budget < 300,000) * (\#\ Dept\ pgs) * (I/O\ cost) \\
&\quad (30\ CPUs) * (50\ pgs/CPU) * t_d \\
&\quad 30 * 50 * t_d \\
&\quad 1,500 * t_d \\
&+ \quad (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ did\ pgs) * t_s \\
&\quad 30 * 99 * 17 * t_s \\
&\quad 30 * 1,683 * t_s \\
&\quad 50,490 * t_s \\
&+ \quad (\#\ CPU\ storing) * (51\ did\ field\ pgs) * (I/O\ cost) \\
&\quad 100 * 51 * t_d \\
&\quad 5,100 * t_d \\
&+ \quad (\#\ CPU\ joining) * (join\ cost)
\end{aligned}
$$

$$
\begin{aligned}
& 100 * (3 * (170 * 1,000) * t_d) \\
& 351,000 * t_d \\
+ \quad & (\#\ CPUs - 1) * (send\ cost) \\
& 99 * t_s \\
= \quad & 1,500 * t_d + 50,490 * t_s + 5,100 * t_d + 351,000 * t_d + 99 * t_s \\
= \quad & 357,600 * t_d + 50,589 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time \quad = \quad & 50 * t_d + 1,683 * t_s + 51 * t_d + 3,510 * t_d + t_s \\
= \quad & 3,611 * t_d + 1,684 * t_s
\end{aligned}
$$

5. Find the average salary over all departments with budget less than 300,000.

   Plan: This query is similar to part 4 above. The difference is that instead of selecting the highest salary during the join and reporting to a chosen processor, each processor retains a running sum of the *sal* field and count of joined tuples. The chosen processor then computes the total sum and divides by the total count to obtain the average. Note that the costs are identical to part 4.

6. Find the salaries of all managers.

   Plan: Employees tuples with an *eid* field equal to a *mgrid* field of a Departments relation may be stored anywhere. Each processor should conduct a complete linear scan of its Departments tuples retaining only the *mgrid* field. Then, each processor sends the result to all others who subsequently store the *mgrid* relation. Next, each processor joins the *mgrid* relation with Employees retaining only the *sal* field of joined tuples.

$$
\begin{aligned}
Total\ Cost \quad = \quad & scan\ Dept\ for\ mgrid\ fields\ at\ all\ CPUs \\
+ \quad & sending\ mgrid\ field\ tuples\ from\ 100\ CPUs\ to\ 99\ CPUs \\
+ \quad & storing\ mgrid\ field\ tuples\ at\ each\ processor \\
+ \quad & joining\ mgrid\ field\ tuples\ with\ Emp\ tuples \\
+ \quad & sending\ local\ results\ to\ chosen\ processor \\
= \quad & (\#\ CPUs\ scanning) * (\#\ Dept\ pgs) * (I/O\ cost) \\
& 100 * 50 * t_d \\
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ did\ pgs) * t_s \\
& 100 * 99 * 17 * t_s \\
& 100 * 1,683 * t_s \\
& 168,300 * t_s \\
+ \quad & (\#\ CPU\ storing) * (1,700\ did\ field\ pgs) * (I/O\ cost) \\
& 100 * 1,700 * t_d \\
& 170,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 100 * (3 * (1,700 * 1,000) * t_d)
\end{aligned}
$$

$$
\begin{aligned}
& 810,000 * t_d \\
+ \quad & (\#\ CPUs - 1) * (send\ cost) \\
& 99 * t_s \\
= \quad & 5,000 * t_d + 168,300 * t_s + 170,000 * t_d + 810,000 * t_d + 99 * t_s \\
= \quad & 985,000 * t_d + 168,399 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time \quad = \quad & 50 * t_d + 1,683 * t_s + 1,700 * t_d + 8,100 * t_d + t_s \\
= \quad & 9,850 * t_d + 1,684 * t_s
\end{aligned}
$$

7. Find the salaries of all managers who manage a department with a budget less than 300,000 and earn more than 100,000.

   Plan: Department tuples with a budget less than 300,000 are located at processors 1 through 30. Employees tuples with a *sal* fields greater than 100,000 are located at processors 11 through 100. Conduct a complete linear scan of all Department tuples retaining only the *mgrid* field of tuples with a *budget* field less than 300,000. Send the new *mgrid* relation to processors 11 through 100. Next, processors 11 through 100 join the new *mgrid* relation with Employees to obtain the desired result. Finally, the answer is forwarded (costlessly) to the chosen output device.

$$
\begin{aligned}
Total\ Cost \quad = \quad & scan\ Dept\ for\ mgrid\ fields\ at\ first\ thirty\ CPUs \\
+ \quad & sending\ mgrid\ field\ tuples\ from\ 10\ CPUs\ to\ 90\ CPUs \\
+ \quad & sending\ mgrid\ field\ tuples\ from\ 20\ CPUs\ to\ 89\ CPUs \\
+ \quad & storing\ mgrid\ field\ tuples\ at\ 90\ CPUs \\
+ \quad & joining\ mgrid\ field\ tuples\ with\ Emp\ tuples\ in\ 90\ CPUs \\
= \quad & (\#\ CPUs\ scanning) * (\#\ Dept\ pgs/CPU) * (I/O\ cost) \\
& 30 * 50 * t_d \\
& 1,500 * t_d \\
+ \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ mgrid\ pgs) * t_s \\
& 10 * 90 * 17 * t_s \\
& 10 * 1,530 * t_s \\
& 15,300 * t_s \\
+ \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ mgrid\ pgs) * t_s \\
& 20 * 89 * 17 * t_s \\
& 20 * 1,513 * t_s \\
& 30,260 * t_s \\
+ \quad & (\#\ CPU\ storing) * (51\ mgrid\ field\ pgs) * (I/O\ cost) \\
& 90 * 51 * t_d \\
& 4,590 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 90 * (3 * (510 + 1,000) * t_d)
\end{aligned}
$$

$$
\begin{aligned}
& 90 * 4,530 * t_d \\
& 407,700 * t_d \\
= \;& 1,500 * t_d + 15,300 * t_s + 30,260 * t_s + 4,590 * t_d + 407,700 * t_d \\
= \;& 413,790 * t_d + 45,560 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 1,530 * t_s + 1,513 * t_s + 51 * t_d + 4,530 * t_d \\
&= 4,631 * t_d + 3,043 * t_s
\end{aligned}
$$

8. Print the eids of all employees, ordered by increasing salaries.
   Plan: Sort the Employees relation at each processor and print it out.

$$
\begin{aligned}
Total\ Cost &= (\#\ CPU sorting) * (sort\ cost) \\
&= 100 * (3 * 1,000 * t_d) \\
&= 300,000 * t_d
\end{aligned}
$$

$$
Elapsed\ Time = 1,000\ pgs * (sort\ cost)
$$

Exercise 21.6 Consider the Employees and Departments relations described in Exercise 21.3. They are now stored in a distributed DBMS with all of Employees stored at Naples and all of Departments stored at Berlin. There are no indexes on these relations. The cost of various operations is as described in Exercise 21.3. Consider the query:

```
SELECT *
FROM   Employees E, Departments D
WHERE  E.eid = D.mgrid
```

The query is posed at Delhi, and you are told that only 1 percent of employees are managers. Find the cost of answering this query using each of the following plans:

1. Compute the query at Naples by shipping Departments to Naples; then ship the result to Delhi.
2. Compute the query at Berlin by shipping Employees to Berlin; then ship the result to Delhi.
3. Compute the query at Delhi by shipping both relations to Delhi.
4. Compute the query at Naples using Bloomjoin; then ship the result to Delhi.
5. Compute the query at Berlin using Bloomjoin; then ship the result to Delhi.
6. Compute the query at Naples using Semijoin; then ship the result to Delhi.
7. Compute the query at Berlin using Semijoin; then ship the result to Delhi.

Answer 21.6 Answer omitted.

Exercise 21.7 Consider your answers in Exercise 21.6. Which plan minimizes shipping costs? Is it necessarily the cheapest plan? Which do you expect to be the cheapest?

Answer 21.7 Answer not available.

Exercise 21.8 Consider the Employees and Departments relations described in Exercise 21.3. They are now stored in a distributed DBMS with 10 sites. The Departments tuples are horizontally partitioned across the 10 sites by *did*, with the same number of tuples assigned to each site and with no particular order to how tuples are assigned to sites. The Employees tuples are similarly partitioned, by *sal* ranges, with $sal \leq 100,000$ assigned to the first site, $100,000 < sal \leq 200,000$ assigned to the second site, and so on. In addition, the partition $sal \leq 100,000$ is frequently accessed and infrequently updated, and it is therefore replicated at every site. No other Employees partition is replicated.

1. Describe the best plan (unless a plan is specified) and give its cost:
   (a) Compute the natural join of Employees and Departments using the strategy of shipping all fragments of the smaller relation to every site containing tuples of the larger relation.
   (b) Find the highest paid employee.
   (c) Find the highest paid employee with salary less than $100,000$.
   (d) Find the highest paid employee with salary greater than $400,000$ and less than $500,000$.
   (e) Find the highest paid employee with salary greater than $450,000$ and less than $550,000$.
   (f) Find the highest paid manager for those departments stored at the query site.
   (g) Find the highest paid manager.
2. Assuming the same data distribution, describe the sites visited and the locks obtained for the following update transactions, assuming that *synchronous* replication is used for the replication of Employees tuples with $sal \leq 100,000$:
   (a) Give employees with salary less than $100,000$ a 10 percent raise, with a maximum salary of $100,000$ (i.e., the raise cannot increase the salary to more than $100,000$).
   (b) Give all employees a 10 percent raise. The conditions of the original partitioning of Employees must still be satisfied after the update.
3. Assuming the same data distribution, describe the sites visited and the locks obtained for the following update transactions, assuming that *asynchronous* replication is used for the replication of Employees tuples with $sal \leq 100,000$.
   (a) For all employees with salary less than $100,000$ give them a 10 percent raise, with a maximum salary of $100,000$.
   (b) Give all employees a 10 percent raise. After the update is completed, the conditions of the original partitioning of Employees must still be satisfied.

Answer 21.8 Answer omitted.

Exercise 21.9 Consider the Employees and Departments relations from Exercise 21.3. You are a DBA dealing with a distributed DBMS, and you need to decide how to distribute these two relations across two sites, Manila and Nairobi. Your DBMS supports only unclustered B+ tree indexes. You have a choice between synchronous and asynchronous replication. For each of the following scenarios, describe how you would distribute them and what indexes you would build at each site. If you feel that you have insufficient information to make a decision, explain briefly.

1. Half the departments are located in Manila, and the other half are in Nairobi. Department information, including that for employees in the department, is changed only at the site where the department is located, but such changes are quite frequent. (Although the location of a department is not included in the Departments schema, this information can be obtained from another table.)
2. Half the departments are located in Manila, and the other half are in Nairobi. Department information, including that for employees in the department, is changed only at the site where the department is located, but such changes are infrequent. Finding the average salary for each department is a frequently asked query.
3. Half the departments are located in Manila, and the other half are in Nairobi. Employees tuples are frequently changed (only) at the site where the corresponding department is located, but the Departments relation is almost never changed. Finding a given employee's manager is a frequently asked query.
4. Half the employees work in Manila, and the other half work in Nairobi. Employees tuples are frequently changed (only) at the site where they work.

Answer 21.9 1. Given that department information is frequently changed only at the site where it is located, horizontal fragmentation based on department location (recall that location is available in another table) will increase performance. Without knowing more about access patterns to employee data, it is impossible to say precisely what should be done with the Employees relation and what indexes would be useful. However, it is likely that given its size, a similar geographically based horizontal fragmentation of Employees along with an index on its *did* field would be useful in coordinating updates to the Departments.

2. Given that department information is infrequently changed only at the site where it is located, replication of Departments at both Manila and Nairobi will have a positive effect. There is insufficient information given to clearly decide on asynchronous vs. synchronous replication. On the one hand, infrequent updates to Departments suggests that the accuracy gains from synchronous replication may outweigh the efficiency loss. Yet on the other hand, the most frequent query is for a departmental salary average, i.e., not for a exact number. Hence the low utility of precision suggests that semi-frequent asynchronous replication may be superior.

   Depending on the access patterns of the departmental average salary queries, Employees may or may not be replicated at both sites. The sheer size of Employees suggests that replication could be very costly. Avoiding it by horizontal fragmentation on the *did* field may prove optimal overall. Even if other accesses were slower, the gains in terms of faster departmental average salary queries might tip the balance. For either replication strategy, indexes on the *did* field in both Employees and Departments would greatly enhance the speed of average salary by department queries.

3. Given that Employees tuples are frequently changed at the home site, horizontal fragmentation is very appealing. Since the results of the queries are for a single employee's manager, i.e., small result relations, there is little incentive for any replication strategy for Employees. Given that Departments almost never changes and is used for exact answer queries, synchronous replication is the best alternative. The slight loss in efficiency is easily won back in the ability to find an employees' manager immediately. The overhead of indexes on each of the key fields, *eid* and *did*, is also easily justified.
4. Given that half of the employees work in Manila, the other half work in Nairobi, and Employees' tuples are frequently changed only where they work; the obvious strategy is to horizontally fragment Employees based on worker's location (assuming this information is available in another table). Indexes on Employees' *did* field for each locations fragment would also speed the frequent accesses necessary for updating the tuples.

Exercise 21.10 Suppose that the Employees relation is stored in Madison and the tuples with $sal \leq 100,000$ are replicated at New York. Consider the following three options for lock management: all locks managed at a *single site*, say, Milwaukee; *primary copy* with Madison being the primary for Employees; and *fully distributed*. For each of the lock management options, explain what locks are set (and at which site) for the following queries. Also state which site the page is read from.

1. A query submitted at Austin wants to read a page containing Employees tuples with $sal \leq 50,000$.
2. A query submitted at Madison wants to read a page containing Employees tuples with $sal \leq 50,000$.
3. A query submitted at New York wants to read a page containing Employees tuples with $sal \leq 50,000$.

Answer 21.10 Answer omitted.

Exercise 21.11 Briefly answer the following questions:

1. Compare the relative merits of centralized and hierarchical deadlock detection in a distributed DBMS.
2. What is a *phantom deadlock*? Give an example.
3. Give an example of a distributed DBMS with three sites such that no two local waits-for graphs reveal a deadlock, yet there is a global deadlock.
4. Consider the following modification to a local waits-for graph: Add a new node T_{ext}, and for every transaction T_i that is waiting for a lock at another site, add the edge $T_i \rightarrow T_{ext}$. Also add an edge $T_{ext} \rightarrow T_i$ if a transaction executing at another site is waiting for T_i to release a lock at this site.
   (a) If there is a cycle in the modified local waits-for graph that does not involve T_{ext}, what can you conclude? If every cycle involves T_{ext}, what can you conclude?

(b) Suppose that every site is assigned a unique integer *site-id*. Whenever the local waits-for graph suggests that there might be a global deadlock, send the local waits-for graph to the site with the next higher site-id. At that site, combine the received graph with the local waits-for graph. If this combined graph does not indicate a deadlock, ship it on to the next site, and so on, until either a deadlock is detected or we are back at the site that originated this round of deadlock detection. Is this scheme guaranteed to find a global deadlock if one exists?

Answer 21.11 1. A centralized deadlock detection scheme is better for a distributed DBMS with uniform access patterns across sites since dead-locks occurring between any two sites are immediately identified. However, this benefit comes at the expense of frequent communications between the central location and every other site.

It is often the case that access patterns are more localized, perhaps by geographic area. Since deadlocks are more likely to occur among sites with frequent communication, the hierarchical scheme will be more efficient in that it checks for deadlocks where they are most likely to occur. In other words, the hierarchical scheme expends deadlock detection efforts in correlation to their probability of occurrence, thus resulting in greater efficiency.

2. A *phantom deadlock* is defined as a falsely identified deadlock resulting from the time delay in sending local waits-for information to a central or parent site. A cycle appearing in the central or parent's global waits-for graph may in actuality have disappeared by the time the graph nodes are received and constructed. The phantom may result in some transactions being killed unnecessarily.

   For example, imagine that transaction T1 at site A is waiting for T2 at site B which is in turn waiting for T1. Then the local waits-for graphs are sent to the central detection site. Meanwhile, transaction T2 aborts for an unrelated reason and T1 no longer waits. Unfortunately for T1, the central site has identified a cycle in the global waits and chooses to kill T1!

3. Imagine three transactions T1, T2, and T3 at sites A, B, and C respectively. Suppose that T1 waits for T2 which in turn waits for T3. Comparing any two graphs in this waits-for-triangle will not reveal the global deadlock.

4. (a) A cycle in the modified waits for graph not involving $T_e x_t$ clearly indicates that the deadlock is internal to the site with the graph. If every cycle involves $T_e x_t$, then there may be a multiple-site or potentially global deadlock.

   (b) The scheme is guaranteed to find a global deadlock provided that the deadlock exists prior to when the first waits-for graph is sent. If this condition is met, then the global deadlock will be uncovered before any node receives a graph containing its own nodes back.

Exercise 21.12 Timestamp-based concurrency control schemes can be used in a distributed DBMS, but we must be able to generate globally unique, monotonically increasing timestamps without a bias in favor of any one site. One approach is to assign timestamps at a single site. Another is to use the local clock time and to append the site-id. A third scheme is to use a counter at each site. Compare these three approaches.

Answer 21.12 Answer omitted.

Exercise 21.13 Consider the multiple-granularity locking protocol described in Chapter 18. In a distributed DBMS the site containing the root object in the hierarchy can become a bottleneck. You hire a database consultant who tells you to modify your protocol to allow only intention locks on the root, and to implicitly grant all possible intention locks to every transaction.

1. Explain why this modification works correctly, in that transactions continue to be able to set locks on desired parts of the hierarchy.
2. Explain how it reduces the demand upon the root.
3. Why isn't this idea included as part of the standard multiple-granularity locking protocol for a centralized DBMS?

Answer 21.13

1. The consultant's suggestion of allowing only intention locks on the root works correctly because it does not prevent any transaction from obtaining a shared or exclusive lock on any sub-structure contained within the root. Recall that to obtain a shared lock, a transaction must first have an intention shared lock and to get an exclusive lock it must first have an intention exclusive lock. The only limitation resulting from the modification is that transactions wishing to modify the entire structure contained within the root must wait to obtain the necessary locks on every child of the root node.
2. The demand upon the root is reduced for two reasons. First, no transaction may greedily occupy the entire structure and must choose the relevant substructure. Second, the implicit granting of all possible intention locks to every transaction requesting access to any structure contained within the root reduces the load on the Lock Manager and the size of the waiting or fairness queue. Transactions need not wait in line for the intention locks. For these reasons, the bottleneck problem will be minimized.
3. This idea is not included as part of the Multiple-Granularity locking protocol for a centralized DBMS, or in general for a distributed DBMS, because it is a custom solution to a specific problem. The standard protocol could not predict which if any root or sub-root structures may become bottlenecks and so as is typical of standards, it opts for the general solution to the given problem.

Exercise 21.14 Briefly answer the following questions:

1. Explain the need for a commit protocol in a distributed DBMS.
2. Describe 2PC. Be sure to explain the need for force-writes.
3. Why are *ack* messages required in 2PC?
4. What are the differences between 2PC and 2PC with Presumed Abort?
5. Give an example execution sequence such that 2PC and 2PC with Presumed Abort generate an identical sequence of actions.
6. Give an example execution sequence such that 2PC and 2PC with Presumed Abort generate different sequences of actions.
7. What is the intuition behind 3PC? What are its pros and cons relative to 2PC?
8. Suppose that a site does not get any response from another site for a long time. Can the first site tell whether the connecting link has failed or the other site has failed? How is such a failure handled?

9. Suppose that the coordinator includes a list of all subordinates in the *prepare* message. If the coordinator fails after sending out either an *abort* or *commit* message, can you suggest a way for active sites to terminate this transaction without waiting for the coordinator to recover? Assume that some but not all of the *abort*/*commit* messages from the coordinator are lost.
10. Suppose that 2PC with Presumed Abort is used as the commit protocol. Explain how the system recovers from failure and deals with a particular transaction T in each of the following cases:
    (a) A subordinate site for T fails before receiving a *prepare* message.
    (b) A subordinate site for T fails after receiving a *prepare* message but before making a decision.
    (c) A subordinate site for T fails after receiving a *prepare* message and force-writing an abort log record but before responding to the *prepare* message.
    (d) A subordinate site for T fails after receiving a *prepare* message and force-writing a prepare log record but before responding to the *prepare* message.
    (e) A subordinate site for T fails after receiving a *prepare* message, force-writing an abort log record, and sending a *no* vote.
    (f) The coordinator site for T fails before sending a *prepare* message.
    (g) The coordinator site for T fails after sending a *prepare* message but before collecting all votes.
    (h) The coordinator site for T fails after writing an *abort* log record but before sending any further messages to its subordinates.
    (i) The coordinator site for T fails after writing a *commit* log record but before sending any further messages to its subordinates.
    (j) The coordinator site for T fails after writing an *end* log record. Is it possible for the recovery process to receive an inquiry about the status of T from a subordinate?

Answer 21.14 Answer omitted.

Exercise 21.15 Consider a heterogeneous distributed DBMS.

1. Define the terms *multidatabase system* and *gateway*.
2. Describe how queries that span multiple sites are executed in a multidatabase system. Explain the role of the gateway with respect to catalog interfaces, query optimization, and query execution.
3. Describe how transactions that update data at multiple sites are executed in a multidatabase system. Explain the role of the gateway with respect to lock management, distributed deadlock detection, Two-Phase Commit, and recovery.
4. Schemas at different sites in a multidatabase system are probably designed independently. This situation can lead to *semantic heterogeneity*; that is, units of measure may differ across sites (e.g., inches versus centimeters), relations containing essentially the same kind of information (e.g., employee salaries and ages) may have slightly different schemas, and so on. What impact does this heterogeneity have on the end user? In particular, comment on the concept of distributed data independence in such a system.

Answer 21.15 1. A *multi-database system* (a.k.a. heterogeneous distributed database system) is defined as a distributed DBMS where sites operate under different DBMS packages or software. A *gateway* is defined as a communication protocol or standard used by two different DBMS packages to transmit information.

2. Queries in a *multi-database system* originate in system designated as the primary DBMS for the given query. Catalog interfaces provide the primary system with the information necessary to optimize and sub-divide the query to the sub-sites where relevant data is located. The primary system sends an optimized SQL query written in a variant of SQL that complies with the *gateway* protocol. After messages are sent back and forth to ensure compliance with locking and commit protocols, the sub-sites re-optimize their queries based on their (presumably more current) catalog information. The sub-sites then process the query and return the resulting tuples to the primary site which assembles them and presents them to the user.

3. Transactions that update data at multiple sites in a heterogeneous distributed DBMS must adhere to agreed upon locking and commit protocols just as in any DBMS with concurrency control and recovery. In any distributed system, a series of messages are associated with updates between sites to guarantee safe atomic transactions. In a heterogeneous system, communication between the different types of DBMS at different sites occurs through the *gateway*. The *gateway* provides communication channels for lock requests and responses, waits-for graphs messages, transmission of recovery logs, and the prepare, yes, no, commit, ack, and abort messages associated with two-phase commit. Given the diversity, frequency, and accuracy requirements of these essential communications it comes as no surprise that efficient gateways have not yet been successfully implemented on a wide scale.

4. The existence of semantic heterogeneity may have a profoundly confusing and/or adverse effect upon the end user. Imagine trying to understand how the average summer temperature in the Sahara desert is only 50 when the measurement is mistakenly assumed to be Fahrenheit. Or even worse, imagine investing your life savings in an security from the London stock exchange because it seems so cheap (if it were really priced in US dollars!). Beyond trivial unit conversions, there may even be different data structures and relational schema at each of the distributed sites.

   In these situations, the goal of distributed data independence, the idea that the user need not know where or how the data is stored, is obviously compromised. A sophisticated DBA could hopefully avoid the misunderstandings above by implicitly converting data to the correct local units. More generally, the DBA could create different global views to mask the underlying inconsistencies between sites. However, a large widely distributed DBMS will cross cultural boundaries and whether for humor or for sorrow, will necessarily instigate some semantic confusion.

CHAPTER 22

Object-Based Databases

Practice Exercises

22.1 A car-rental company maintains a database for all vehicles in its current fleet. For all vehicles, it includes the vehicle identification number, license number, manufacturer, model, date of purchase, and color. Special data are included for certain types of vehicles:

- Trucks: cargo capacity.
- Sports cars: horsepower, renter age requirement.
- Vans: number of passengers.
- Off-road vehicles: ground clearance, drivetrain (four- or two-wheel drive).

Construct an SQL schema definition for this database. Use inheritance where appropriate.

Answer: For this problem, we use table inheritance. We assume that **MyDate**, **Color** and **DriveTrainType** are pre-defined types.

create type *Vehicle*
(*vehicle_id* **integer**,
license_number **char(15)**,
manufacturer **char(30)**,
model **char(30)**,
purchase_date **MyDate**,
color **Color**)

create table *vehicle* **of type** *Vehicle*

create table *truck*
(*cargo_capacity* **integer**)
under *vehicle*

create table *sportsCar*

```
    (horsepower integer
     renter_age_requirement integer)
    under vehicle

create table van
    (num_passengers integer)
    under vehicle

create table offRoadVehicle
    (ground_clearance real
     driveTrain DriveTrainType)
    under vehicle
```

22.2 Consider a database schema with a relation *Emp* whose attributes are as shown below, with types specified for multivalued attributes.

Emp = (ename, ChildrenSet **multiset***(Children), SkillSet* **multiset***(Skills))*
Children = (name, birthday)
Skills = (type, ExamSet **setof***(Exams))*
Exams = (year, city)

a. Define the above schema in SQL, with appropriate types for each attribute.

b. Using the above schema, write the following queries in SQL.

   i. Find the names of all employees who have a child born on or after January 1, 2000.

   ii. Find those employees who took an examination for the skill type "typing" in the city "Dayton".

   iii. List all skill types in the relation *Emp*.

Answer:

a. No Answer.

b. Queries in SQL.

   i. Program:

```
select ename
from emp as e, e.ChildrenSet as c
where 'March' in
        (select birthday.month
         from c
        )
```

   ii. Program:

```
select e.ename
from emp as e, e.SkillSet as s, s.ExamSet as x
where s.type = 'typing' and x.city = 'Dayton'
```

iii. Program:

```
select distinct s.type
from emp as e, e.SkillSet as s
```

22.3 Consider the E-R diagram in Figure 22.5, which contains composite, multivalued, and derived attributes.

a. Give an SQL schema definition corresponding to the E-R diagram.

b. Give constructors for each of the structured types defined above.

Answer:

a. The corresponding SQL:1999 schema definition is given below. Note that the derived attribute *age* has been translated into a method.

```
create type Name
    (first_name varchar(15),
     middle_initial char,
     last_name varchar(15))
create type Street
    (street_name varchar(15),
     street_number varchar(4),
     apartment_number varchar(7))
create type Address
    (street Street,
     city varchar(15),
     state varchar(15),
     zip_code char(6))
create table customer
    (name Name,
     customer_id varchar(10),
     address Adress,
     phones  char(7)  array[10],
     dob date)
method integer age()
```

b.
```
create function Name (f varchar(15), m char, l varchar(15))
returns Name
begin
    set first_name = f;
    set middle_initial = m;
    set last_name = l;
end
create function Street (sname varchar(15), sno varchar(4), ano varchar(7))
```

```
returns Street
begin
    set street_name = sname;
    set street_number = sno;
    set apartment_number =ano;
end
create function Address (s Street, c varchar(15), sta varchar(15), zip varchar(6))
returns Address
begin
    set street = s;
    set city = c;
    set state =sta;
    set zip_code =zip;
end
```

22.4 Consider the relational schema shown in Figure 22.6.

a. Give a schema definition in SQLcorresponding to the relational schema, but using references to express foreign-key relationships.

b. Write each of the queries given in Exercise 6.13 on the above schema, using SQL.

Answer:

a. The schema definition is given below. Note that backward references can be addedbut they are not so important as in OODBS because queries can be written in SQL and joins can take care of integrity constraints.

```
create type Employee
    (person_name varchar(30),
     street varchar(15),
     city varchar(15))
create type Company
    (company_name varchar(15),
    (city varchar(15))
create table employee of Employee
create table company of Company
create type Works
    (person ref(Employee) scope employee,
    comp ref(Company) scope company,
    salary int)
create table works of Works
create type Manages
    (person ref(Employee) scope employee,
    (manager ref(Employee) scope employee)
create table manages of Manages
```

b. i. **select** *comp*− >*name*

from *works*
group by *comp*
having count(*person*) $\geq$ **all(select count(**$person$**)**
from *works*
group by *comp*)

ii. **select** $comp->name$
from *works*
group by *comp*
having sum(*salary*) $\leq$ **all(select sum(**$salary$**)**
from *works*
group by *comp*)

iii. **select** $comp->name$
from *works*
group by *comp*
having avg(*salary*) $>$ **(select avg(**$salary$**)**
from *works*
where $comp->company\_name$="First Bank Corporation")

22.5 Suppose that you have been hired as a consultant to choose a database system for your client's application. For each of the following applications, state what type of database system (relational, persistent programming language–based OODB, object relational; do not specify a commercial product) you would recommend. Justify your recommendation.

a. A computer-aided design system for a manufacturer of airplanes.

b. A system to track contributions made to candidates for public office.

c. An information system to support the making of movies.

Answer:

a. A computer-aided design system for a manufacturer of airplanes:
An OODB system would be suitable for this. That is because CAD requires complex data types, and being computation oriented, CAD tools are typically used in a programming language environment needing to access the database.

b. A system to track contributions made to candidates for public office:
A relational system would be apt for this, as data types are expected to be simple, and a powerful querying mechanism is essential.

c. An information system to support the making of movies:
Here there will be extensive use of multimedia and other complex data types. But queries are probably simple, and thus an object relational system is suitable.

22.6 How does the concept of an object in the object-oriented model differ from the concept of an entity in the entity-relationship model?

Answer: An entity is simply a collection of variables or data items. An object is an encapsulation of data as well as the methods (code) to operate on the data. The data members of an object are directly visible only to its methods. The outside world can gain access to the object's data only by passing pre-defined messages to it, and these messages are implemented by the methods.

1. Why we need to consider optimizing queries for parallel execution?
2. Define and describe Distributed Timestamp-Based Protocols. Explain in brief distributed validation.
3. Define and describe types of skew. How to handling Skew in Range-Partitioning
4. What are the similarities and differences between parallel and distributed database management systems?
5. Describe Two-Phase Commit Protocol in distributed database system

6. Explain in brief Array and Multiset Types in SQL.

7. Consider a database schema with a relation Emp whose attributes are as shown below, with types specified for multivalued attributes.

Emp = (ename, ChildrenSet multiset(Children), SkillSet multiset(Skills))
Children = (name, birthday)
Skills = (type, ExamSet setof(Exams))
Exams = (year, city)

a) Define the above schema in SQL, with appropriate types for each attribute.

b) Using the above schema, write the following queries in SQL.

i. Find the names of all employees who have a child born on or after January 1, 2000.

ii. Find those employees who took an examination for the skill type "typing" in the city "Dayton".

iii. List all skill types in the relation Emp.

8.

Unit #3: XML Databases

Q #9) What is the difference between XML and HTML?
Answer:

XML	HTML
XML consists of user defined tags.	HTML consists of pre-defined tags.
XML is used to store and transform data.	HTML is used for designing a web pag
XML is content driven.	HTML is format driven.
XML is case sensitive.	HTML is not case sensitive.
XML requires end tag for well formatted document.	HTML does not require an end tag.

Q #10) What are the benefits of XML?
Answer: The benefits of XML are as follows:

- **Simplicity:** XML is simple to read and understand.
- **Availability:** XML can be created using any text editor.
- **Flexibility:** XML doesn't have any fixed tags, hence user-defined tags can also be used.

Q #11) What importance does XSLT hold in XML?
Answer: XSLT stands for Extensible Style sheet Language Transformation. It is used to transform an XML document to HTML before it is displayed to any browser.
Q #12) What is XQuery?
Answer: XQuery is used to fetch data from the XML file, which is the SQL database.
Q #13) What is Xlink in XML?
Answer: Xlink used in XML files, are the standard way of creating hyperlinks in XML files.
Q #14) What is Xpointer in XML?
Answer: Xpointer in XML allows hyperlinks to point to more specific parts of the XML documents or files.
Q #15) What is XML signature/encryption?
Answer: It defines the processing rules and syntax for encrypting and creating digital signatures on XML.
Q #16) What is DTD in XML?
Answer: DTD stands for Document Type Definition, which describes a document written in XML. XML declaration syntax is defined in DTD. Naming convention rules of different types of elements are also defined in DTD.
Q #17) What is DOM? What is it used for?
Answer: DOM stands for the Document Object Model. It is an API, Application Programming Interface that allows navigation through objects. Documents are treated as objects. DOM documents are generated by the user or created by a parser.
Q #18) What is the main disadvantage of DOM?
Answer: The main disadvantage is that a large portion of memory is consumed by DOM.
Q #19) What does SOAP stand for?
Answer: SOAP is a Simple Object Access Protocol.
Q #20) What is the relation of SOAP with XML?

Answer: SOAP uses XML to define a protocol for the exchange of information in distributed computing environments.

Q #21) What are the three components in SOAP?

Answer: It consists of an envelope, a set of encoding rules, and a convention for representing remote procedure calls.

Q #22) What is XML parser function?

Answer: It is used to convert an XML file or document into the XML DOM object which is usually written in JavaScript.

Q #23) What is an XML schema?

Answer: XML schema provides definition of an XML document.

It comprises of:

- Attributes and elements.
- Child elements.
- The data type of elements.
- Order of elements and attributes.

Q #24) What is CDATA in XML?

Answer: CDATA stands for character data. Characters like '<' and '>' are not allowed in XML. CDATA starts with <! CDATA ["and end with"]>. CDATA is an unparsed character data that cannot be parsed by the XML parser.

Q #25) How are comments used in XML?

Answer: Comments are displayed as <!—comment–> . It is similar to HTML. It can be used for a single line or multiple lines.

Q #26) What is the usage of XML in development?

Answer: XML has multiple usages as shown below:

- XML is used for flat files and databases.
- It is used to store data and transport data on the Internet.
- It can generate different dynamic data using style sheets.
- XML is used to develop database-driven websites.
- It is used to store data for eCommerce websites.

Q #33) Describe XPath.

Answer: XPath can be described as follows:

1. XPath is a W3C recommendation.
2. It is the syntax for defining parts of an XML document.
3. It uses path expressions to navigate in the XML documents.
4. XPath contains a standard function library.
5. XPath is a major element of the XSLT standard.
6. It is used to navigate through attributes and elements in an XML document.

Q #34) Provide an example of XML.

Answer:

```
<? XML version="1.0" encoding = "UTF-8"?>
<FurnitureStore>
<Furniture category="Table">
<Title lang="en"> Sale for today</Title>
<Type> Laptop table</Type>
<Year>2008</Year>
<Price>500</Price>
</FurnitureStore>
```

Q #35) What are well-formed XML documents?

Answer: Well-formed XML documents have the following features:

1. An XML document must have a root element.
2. XML tags are case sensitive.
3. XML elements should be properly nested.

4. XML values should be properly quoted.
5. XML tags should be closed properly.

Q #36) What are XML attributes? Explain with an example.

Answer: XML attribute values should always be quoted. Single or double quotes can be used in XML.

For example:

- <Person degree = “PHD”>
- <Person name = ‘Jacob’>

Q #37) Write a code for XML attribute and element.

Answer:

```
<Person location = “India”>
<Statename>Maharashtra</Statename>
<Cityname>Mumbai</Cityname>
</Person>
```

```
     <Person>
     <Location> India </Location>
     <Statename>Maharashtra</Statename>
     <Cityname>Mumbai</Cityname>
     </Person>
```

In the first element, location is an attribute. In last, location is an element. The user can choose the attribute or element.

Q #38) Can XML files be viewed in browsers?

Answer: Yes, the XML file can be viewed in all known browsers. They are not displayed as HTML pages.

Q #39) What is XML Httprequest? What are its advantages?

Answer: All modern browsers have a built-in XML Httprequest object to request for data from a server.

Its advantages are as follows:

- Updating a web page without reloading the page.
- Request data from a server
- Receive data from a server after the page has been loaded.
- Send data to a server in the background.

Q #40) Example of HttpRequest.

Answer:

```
var xhttp= newXML Httprequest();
Xhttp.onreadystatechange=function();
{ If this.readystate==4&& this.status==200)
        { Action to be performed when document is ready;
Document.getelementbyID(“Demo”)
Innerhtml=xhttp.responseText;}};
```

Q #41) What is XML element?

Answer: The XML element contains start tag, end tag, and values.

For Example:

- <City> Pune </City>
- <Price> 400.00 </Price>

XML element with no value is said to be empty like <element> </element>

Q #42) What are XML naming rules?

Answer: Naming rules are:

- Element names must start with a letter or underscore.
- Element names are case sensitive.
- Element names cannot start with the letters XML.
- Element names can contain letters, digits, hyphens, underscore, and periods.

- Element names cannot contain spaces.

Q #43) What is SAX in XML?

Answer: SAX stands for Simple API for XML. It is a sequential access parser.
It provides a mechanism of reading data from an XML document. It is said to be an alternative to DOM. DOM operates on the documents as a whole, SAX parsers operate on each piece of the XML document sequentially.

SAX consumes less memory. It cannot be used to write an XML document.

Q #44) What is XSNL?

Answer: XSNL stands for XML Search Neutral Language. This language acts between the meta-search interface and the targeted system.

Q #45) What is the difference between a simple element and a complex element?

Answer: Simple elements cannot be left empty. It contains fewer attributes, child elements, etc. Simple elements are text-based elements. Complex elements can contain sub-elements, empty elements, etc. The complex element can hold multiple attributes and elements.

Set#2

3. When would I use XML instead of SQL?

Ans: SQL is good for tabular data, or information that fits neatly into rows and columns. XML is ideal for hierarchical data or data that has multiple levels of varying sizes. SQL is useful for storing and searching data. XML is useful for both conveying and formatting data.

Instead of developing a whole database, you might utilise XML. However, I would recommend that you consider the type and amount of data you want to keep, as well as your justifications for not using a database. The relational elements of a database, such as MySql or SQL, are one of the benefits of using one. The database server's ability to execute a reasonable amount of work, as well as the fact that this is a typical method of storing data in the tech world.

Nowadays, there are several alternatives to using a full-scale database, one of which is XML.
You can also store data as a JSON object if you're using Javascript, or as an array in whatever programming language, you're using. There are NoSql databases that use JSON to store data as a single large array of key => value pairs that can be nested for quite complex data structures. If you're using a framework like React, you can also use Flux or Redux to store data in a Javascript structure known as a Store.

As a result, there are a number of alternatives to using a traditional database package like MySql or SQL. However, as I previously stated, I would advise you to carefully consider the reasons for not wanting to use a database, as well as whether the data you intend to store is suitable for storage in one of the alternative options.

4. List some features of XML?

Ans:

- for managing data with a complex structure or data that is out of the ordinary.
- Markup language is used to describe data.
- Description of text data
- Format that is both human and computer-friendly
- Deals with data in a tree structure with only one root element.
- Long-term data storage and reusability are both excellent.

5. What is XML DOM?

Ans: The document object model is abbreviated as DOM. It specifies how to access and manipulate documents in a consistent manner. The Document Object Model (DOM) is an HTML and XML document programming interface. It specifies how documents are retrieved and changed, as well as their logical structure.

One of the main goals of the Document Object Model as a W3C definition is to provide a common programming interface that can be utilised in a wide range of settings and applications. Any programming language can use the Document Object Model. The XML Document Object Model (DOM) specifies a standard for accessing and manipulating XML documents.

6. How XML is different from HTML?

Ans: HTML and XML vary in that HTML displays data and describes the structure of a webpage, while XML stores and transfers data. HTML is a predetermined language with its own consequences, but XML is a standard language that can define additional computer languages.

7. What is XML Schema?

Ans: XML Schema Definition(XSD) is another name for XML Schema . It describes and validates the structure and content of XML data. The elements, attributes, and data types are defined by the XML schema. Namespaces are supported by Schema elements. It is analogous to a database schema, which describes the data in a database.

10. What is DTD?

Ans: A Document Type Definition (DTD) describes a document's tree structure as well as some information about its data. It is a set of markup assertions that define a type of document for the SGML family, such as GML, SGML, HTML, and XML.

A DTD can be declared inline or as an external recommendation within an XML document. The DTD specifies how many times a node should appear and how their child nodes should be ordered.

PCDATA and CDATA are the two data types.

- PCDATA is character data that has been parsed.
- CDATA is character data that is not typically parsed.

1. **Consider the following recursive DTD:**

<!DOCTYPE parts [

<!ELEMENT part (name, subpartinfo*)>

<!ELEMENT subpartinfo (part, quantity)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT quantity (#PCDATA)>

] >

a. Give a small example of data corresponding to this DTD.

b. Show how to map this DTD to a relational schema. You can assume that part names are unique; that is, wherever a part appears, its subpart structure will be the same.

c. Create a schema in XML Schema corresponding to this DTD.

2. **What is XML query? Write all application of XML in detail.**

12.

Unit #4: Mobile Databases

1. **What is a Mobile Database System (MDS)? Explain.**
2. **What is mobile and intermittent connectivity? Explain**
3. **Personal Communication System (PCS) with neat sketch**
4. **Explain various problems with cellular structure with neat sketches**
5. **Explain Handoff types with reference to link transfer**
6. **With respect to PCS, explain with neat sketch various steps in Location Management**
7. Explain with neat sketch **A Reference Architecture (Client-Server model) of Mobile database system (MDS)**
8. What are **MDS Data Management Issues? Explain each one in brief**
9. What are various **Mobile Transaction Models? Explain each one in brief**
10. Describe in brief frequency reuse by personal communication system.
11. Define and Describe the architecture of Mobile Database System. What are the issues of Mobile Database System.
12. Define Handoff in personal communication system. Explain Handoff detection strategies in detail.
13. Explain in brief transaction management of Mobile Database System.
14.

Unit #5 & 6

1. **What is an Intelligent Database? What are the Features of an Ideal Intelligent DB?**
2. **What is a Multimedia DBMS? What are the requirements of Multimedia DBMS?**
3. **Expalin Relational Calculi**
4. **Explain in brief System Architecture of Petrographer.**
5. **What are the major issues in Multimedia DBMS.**
6. **Describe Recursive Queries in SQL.**

7. **What is Multimedia database? And What are Content of Multimedia Database management system?**

Multimedia database is the collection of interrelated multimedia data that includes text, graphics (sketches, drawings), images, animations, video, audio etc and have vast amounts of multisource multimedia data. The framework that manages different types of multimedia data which can be stored, delivered and utilized in different ways is known as multimedia database management system. There are three classes of the multimedia database which includes static media, dynamic media and dimensional media.
Content of Multimedia Database management system :

1. **Media data –** The actual data representing an object.
2. **Media format data –** Information such as sampling rate, resolution, encoding scheme etc. about the format of the media data after it goes through the acquisition, processing and encoding phase.
3. **Media keyword data –** Keywords description relating to the generation of data. It is also known as content descriptive data. Example: date, time and place of recording.
4. **Media feature data –** Content dependent data such as the distribution of colors, kinds of texture and different shapes present in data.

3. **What are various Areas where multimedia database is applied?**

- **Documents and record management :** Industries and businesses that keep detailed records and variety of documents. Example: Insurance claim record.
- **Knowledge dissemination :** Multimedia database is a very effective tool for knowledge dissemination in terms of providing several resources. Example: Electronic books.

- **Education and training :** Computer-aided learning materials can be designed using multimedia sources which are nowadays very popular sources of learning. Example: Digital libraries.
- Marketing, advertising, retailing, entertainment and travel. Example: a virtual tour of cities.
- **Real-time control and monitoring :** Coupled with active database technology, multimedia presentation of information can be very effective means for monitoring and controlling complex tasks Example: Manufacturing operation control.

4. What are various challenges to multimedia databases? Explain.

There are still many challenges to multimedia databases, some of which are:

1. **Modelling –** Working in this area can improve database versus information retrieval techniques thus, documents constitute a specialized area and deserve special consideration.
2. **Design –** The conceptual, logical and physical design of multimedia databases has not yet been addressed fully as performance and tuning issues at each level are far more complex as they consist of a variety of formats like JPEG, GIF, PNG, MPEG which is not easy to convert from one form to another.
3. **Storage –** Storage of multimedia database on any standard disk presents the problem of representation, compression, mapping to device hierarchies, archiving and buffering during input-output operation. In DBMS, a ”BLOB”(Binary Large Object) facility allows untyped bitmaps to be stored and retrieved.
4. **Performance –** For an application involving video playback or audio-video synchronization, physical limitations dominate. The use of parallel processing may alleviate some problems but such techniques are not yet fully developed. Apart from this multimedia database consume a lot of processing time as well as bandwidth.
5. **Queries and retrieval –**For multimedia data like images, video, audio accessing data through query opens up many issues like efficient query formulation, query execution and optimization which need to be worked upon.

5. Difference Between Multimedia and Hypermedia

Multimedia integrates different forms of content such as text, audio, video, graphics, etc., to form a single presentation. We come across so many different forms of multimedia element on almost every webpage or App that we open. Multimedia is definitely heavy than the traditional static text-based content, but any form of multimedia can immediately attract the users' attention and convey the message quickly.

Hypermedia is a mix of hypertext with media such as graphics, sounds, and animations. It can be understood as the improved version of hypertext.

Read through this article to find out more about Multimedia and Hypermedia and how they are different from each other.

What is Multimedia?

In contrast to conventional mass media such as printed material or audio recordings, multimedia is a type of communication that integrates multiple content formats such as text, audio, pictures, animations, or video into a single presentation. Video podcasts, audio slideshows, and animated videos are all forms of multimedia.

Multimedia components provide us with high-quality images, animations, sounds, and text information that have a direct influence on the user's brain. We can also edit all these forms of multimedia.

What is Hypermedia?

Hypermedia is the next generation of hypertext, containing a variety of media such as images, text, audio, video, and moving visuals. Both hypermedia and hypertext have a structure that is comparable.

Hypermedia includes much more sophisticated capabilities such as webpage clickable connections. The most frequent hypermedia link is an image link that leads to another website.

Hypermedia has a wide range of applications from problem solving and qualitative research to electronic studying and sophisticated learning. The Aspen Movie Map was, perhaps, the first hypermedia creation. HyperCard by Bill Atkinson popularized hypermedia writing, and a number of literary hypertext and hypertext works, both fiction and nonfiction, illustrated the promise of connections.

The majority of current hypermedia is provided via electronic pages from a wide range of technologies, including media players, web browsers, and standalone programs.

Difference between Multimedia and Hypermedia

The following table highlights the major differences between Multimedia and Hypermedia −

Comparison	Multimedia	Hypermedia
Basic	Multimedia reflects the different ways in which information may be represented.	It is a hypertext extension rather than a text-based system.
Types available	There are both linear and non-linear options available.	Only non-linear options are available.
Relation	Hypermedia is created when it is combined with hypertext.	It represents information that is mix of hypertext and multimedia.

Based on	It mostly relies on interactivity and interaction.	It's utilized for crossreferencing as well as inter-connectivity between components.
Requirements of hardware	Multimedia necessitates its own distribution mechanism, which is referred to as a multimedia delivery system.	To improve capability, it gives clickable connections.
Contents	Multimedia is a mix of media and content that saves data in some manner across many devices.	Hypermedia contains more contrasted characters and it is commonly utilized in non-linear data analysis.