CHAPTER 25

Advanced Data Types and New Applications

Practice Exercises

25.1 What are the two types of time, and how are they different? Why does it make sense to have both types of time associated with a tuple?

Answer: A temporal database models the changing states of some aspects of the real world. The time intervals related to the data stored in a temporal database may be of two types - *valid time* and *transaction time*. The valid time for a fact is the set of intervals during which the fact is true in the real world. The transaction time for a data object is the set of time intervals during which this object is part of the physical database. Only the transaction time is system dependent and is generated by the database system.

Suppose we consider our sample bank database to be bitemporal. Only the concept of valid time allows the system to answer queries such as - "What was Smith's balance two days ago?". On the other hand, queries such as - "What did we record as Smith's balance two days ago?" can be answered based on the transaction time. The difference between the two times is important. For example, suppose, three days ago the teller made a mistake in entering Smith's balance and corrected the error only yesterday. This error means that there is a difference between the results of the two queries (if both of them are executed today).

25.2 Suppose you have a relation containing the x, y coordinates and names of restaurants. Suppose also that the only queries that will be asked are of the following form: The query specifies a point, and asks if there is a restaurant exactly at that point. Which type of index would be preferable, R-tree or B-tree? Why?

Answer: The given query is not a range query, since it requires only searching for a point. This query can be efficiently answered by a B-tree index on the pair of attributes (x, y).

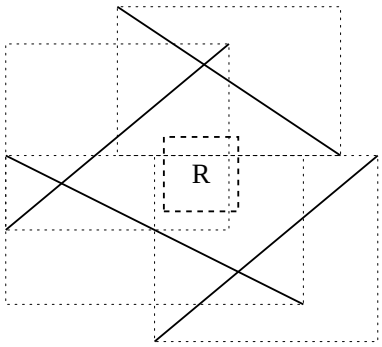
25.3 Suppose you have a spatial database that supports region queries (with circular regions) but not nearest-neighbor queries. Describe an algorithm to find the nearest neighbor by making use of multiple region queries.

Answer: Suppose that we want to search for the nearest neighbor of a point P in a database of points in the plane. The idea is to issue multiple region queries centered at P. Each region query covers a larger area of points than the previous query. The procedure stops when the result of a region query is non-empty. The distance from P to each point within this region is calculated and the set of points at the smallest distance is reported.

25.4 Suppose you want to store line segments in an R-tree. If a line segment is not parallel to the axes, the bounding box for it can be large, containing a large empty area.

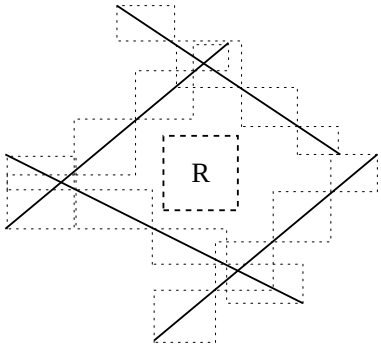
- Describe the effect on performance of having large bounding boxes on queries that ask for line segments intersecting a given region.
- Briefly describe a technique to improve performance for such queries and give an example of its benefit. Hint: You can divide segments into smaller pieces.

Answer: Large bounding boxes tend to overlap even where the region of overlap does not contain any information. The following figure:



shows a region R within which we have to locate a segment. Note that even though none of the four segments lies in R, due to the large bounding boxes, we have to check each of the four bounding boxes to confirm this.

A significant improvement is observed in the follwoing figure:



where each segment is split into multiple pieces, each with its own bounding box. In the second case, the box R is not part of the boxes indexed by the R-tree. In general, dividing a segment into smaller pieces causes the bounding boxes to be smaller and less wasteful of area.

25.5 Give a recursive procedure to efficiently compute the spatial join of two relations with R-tree indices. (Hint: Use bounding boxes to check if leaf entries under a pair of internal nodes may intersect.)

Answer: Following is a recursive procedure for computing spatial join of two R-trees.

```
SpJoin(node n1, node n2)
begin
    if(the bounding boxes of n1 and n2 do not intersect)
        return;
    if(both n1 and n2 are leaves)
        output all pairs of entries (e1, e2) such that
            e1 ∈ n1 and e2 ∈ n2, and e1 and e2 overlap;
    if(n1 is not a leaf)
        NS1 = set of children of n1;
    else
        NS1 = { n1 };
    if(n1 is not a leaf)
        NS1 = set of children of n1;
    else
        NS1 = { n1 };
    for each ns1 in NS1 and ns2 in NS2;
        SpJoin(ns1, ns2);
end
```

25.6 Describe how the ideas behind the RAID organization (Section 10.3) can be used in a broadcast-data environment, where there may occasionally be noise that prevents reception of part of the data being transmitted.

Answer: The concepts of RAID can be used to improve reliability of the broadcast of data over wireless systems. Each block of data that is to be broadcast is split into *units* of equal size. A checksum value is calculated for each unit and appended to the unit. Now, parity data for these units is calculated. A checksum for the parity data is appended to it to form a parity unit. Both the data units and the parity unit are then broadcast one after the other as a single transmission.

On reception of the broadcast, the receiver uses the checksums to verify whether each unit is received without error. If one unit is found to be in error, it can be reconstructed from the other units.

The size of a unit must be chosen carefully. Small units not only require more checksums to be computed, but the chance that a burst of noise corrupts more than one unit is also higher. The problem with using large units is that the probability of noise affecting a unit increases; thus there is a tradeoff to be made.

25.7 Define a model of repeatedly broadcast data in which the broadcast medium is modeled as a virtual disk. Describe how access time and data-transfer rate for this virtual disk differ from the corresponding values for a typical hard disk.

Answer: We can distinguish two models of broadcast data. In the case of a pure broadcast medium, where the receiver cannot communicate with the broadcaster, the broadcaster transmits data with periodic cycles of retransmission of the entire data, so that new receivers can catch up with all the broadcast information. Thus, the data is broadcast in a continuous cycle. This period of the cycle can be considered akin to the worst case rotational latency in a disk drive. There is no concept of seek time here. The value for the cycle latency depends on the application, but is likely to be at least of the order of seconds, which is much higher than the latency in a disk drive.

In an alternative model, the receiver can send requests back to the broadcaster. In this model, we can also add an equivalent of disk access latency, between the receiver sending a request, and the broadcaster receiving the request and responding to it. The latency is a function of the volume of requests and the bandwidth of the broadcast medium. Further, queries may get satisfied without even sending a request, since the broadcaster happened to send the data either in a cycle or based on some other receivers request. Regardless, latency is likely to be at least of the order of seconds, again much higher than the corresponding values for a hard disk.

A typical hard disk can transfer data at the rate of 1 to 5 megabytes per second. In contrast, the bandwidth of a broadcast channel is typically only a few kilobytes per second. Total latency is likely to be of the order of seconds to hundreds or even thousands of seconds, compared to a few milliseconds for a hard disk.

25.8 Consider a database of documents in which all documents are kept in a central database. Copies of some documents are kept on mobile

computers. Suppose that mobile computer A updates a copy of document 1 while it is disconnected, and, at the same time, mobile computer B updates a copy of document 2 while it is disconnected. Show how the version-vector scheme can ensure proper updating of the central database and mobile computers when a mobile computer reconnects.

Answer: Let C be the computer onto which the central database is loaded. Each mobile computer (host) i stores, with its copy of each document d, a version-vector – that is a set of version numbers $V_{d,i,j}$, with one entry for each other host j that stores a copy of the document d, which it could possibly update.

Host A updates document 1 while it is disconnected from C. Thus, according to the version vector scheme, the version number $V_{1,A,A}$ is incremented by one.

Now, suppose host A re-connects to C. This pair exchanges version-vectors and finds that the version number $V_{1,A,A}$ is greater than $V_{1,C,A}$ by 1, (assuming that the copy of document 1 stored host A was updated most recently only by host A). Following the version-vector scheme, the version of document 1 at C is updated and the change is reflected by an increment in the version number $V_{1,C,A}$. Note that these are the only changes made by either host.

Similarly, when host B connects to host C, they exchange version-vectors, and host B finds that $V_{1,B,A}$ is one less than $V_{1,C,A}$. Thus, the version number $V_{1,B,A}$ is incremented by one, and the copy of document 1 at host B is updated.

Thus, we see that the version-vector scheme ensures proper updating of the central database for the case just considered. This argument can be very easily generalized for the case where multiple off-line updates are made to copies of document 1 at host A as well as host B and host C. The argument for off-line updates to document 2 is similar.