28 ADDITIONAL TOPICS

> This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.
>
> —Winston Churchill

In this book we have concentrated on relational database systems and discussed several fundamental issues in detail. However, our coverage of the database area, and indeed even the relational database area, is far from exhaustive. In this chapter we look briefly at several topics that we did not cover, with the goal of giving the reader some perspective and indicating directions for further study.

We begin with a discussion of advanced transaction processing concepts in Section 28.1. We discuss integrated access to data from multiple databases in Section 28.2, and touch upon mobile applications that connect to databases in Section 28.3. We consider the impact of increasingly larger main memory sizes in Section 28.4. We discuss multimedia databases in Section 28.5, geographic information systems in Section 28.6, and sequence data in Section 28.7. We conclude with a look at information visualization in Section 28.8.

The applications covered in this chapter are pushing the limits of currently available database technology and are motivating the development of new techniques. As even our brief coverage indicates, there is much work ahead for the database field!

28.1 ADVANCED TRANSACTION PROCESSING

The concept of a transaction has wide applicability for a variety of distributed computing tasks, such as airline reservations, inventory management, and electronic commerce.

28.1.1 Transaction Processing Monitors

Complex applications are often built on top of several **resource managers**, such as database management systems, operating systems, user interfaces, and messaging software. A **transaction processing monitor** glues together the services of several resource managers and provides application programmers with a uniform interface for

developing transactions with the ACID properties. In addition to providing a uniform interface to the services of different resource managers, a TP monitor also routes transactions to the appropriate resource managers. Finally, a TP monitor ensures that an application behaves as a transaction by implementing concurrency control, logging, and recovery functions, and by exploiting the transaction processing capabilities of the underlying resource managers.

TP monitors are used in environments where applications require advanced features such as access to multiple resource managers; sophisticated request routing (also called **workflow management**); assigning priorities to transactions and doing priority-based load-balancing across servers; and so on. A DBMS provides many of the functions supported by a TP monitor in addition to processing queries and database updates efficiently. A DBMS is appropriate for environments where the wealth of transaction management capabilities provided by a TP monitor is not necessary and, in particular, where very high scalability (with respect to transaction processing activity) and interoperability are not essential.

The transaction processing capabilities of database systems are improving continually. For example, many vendors offer distributed DBMS products today in which a transaction can execute across several resource managers, each of which is a DBMS. Currently, all the DBMSs must be from the same vendor; however, as transaction-oriented services from different vendors become more standardized, distributed, heterogeneous DBMSs should become available. Eventually, perhaps, the functions of current TP monitors will also be available in many DBMSs; for now, TP monitors provide essential infrastructure for high-end transaction processing environments.

28.1.2 New Transaction Models

Consider an application such as computer-aided design, in which users retrieve large design objects from a database and interactively analyze and modify them. Each transaction takes a long time—minutes or even hours, whereas the TPC benchmark transactions take under a millisecond—and holding locks this long affects performance. Further, if a crash occurs, undoing an active transaction completely is unsatisfactory, since considerable user effort may be lost. Ideally we want to be able to restore most of the actions of an active transaction and resume execution. Finally, if several users are concurrently developing a design, they may want to see changes being made by others without waiting until the end of the transaction that changes the data.

To address the needs of long-duration activities, several refinements of the transaction concept have been proposed. The basic idea is to treat each transaction as a collection of related **subtransactions**. Subtransactions can acquire locks, and the changes made by a subtransaction become visible to other transactions after the subtransaction ends (and before the main transaction of which it is a part commits). In **multilevel trans-**

actions, locks held by a subtransaction are released when the subtransaction ends. In **nested transactions**, locks held by a subtransaction are assigned to the parent (sub)transaction when the subtransaction ends. These refinements to the transaction concept have a significant effect on concurrency control and recovery algorithms.

28.1.3 Real-Time DBMSs

Some transactions must be executed within a user-specified **deadline**. A **hard deadline** means the value of the transaction is zero after the deadline. For example, in a DBMS designed to record bets on horse races, a transaction placing a bet is worthless once the race begins. Such a transaction should not be executed; the bet should not be placed. A **soft deadline** means the value of the transaction decreases after the deadline, eventually going to zero. For example, in a DBMS designed to monitor some activity (e.g., a complex reactor), a transaction that looks up the current reading of a sensor must be executed within a short time, say, one second. The longer it takes to execute the transaction, the less useful the reading becomes. In a real-time DBMS, the goal is to maximize the value of executed transactions, and the DBMS must prioritize transactions, taking their deadlines into account.

28.2 INTEGRATED ACCESS TO MULTIPLE DATA SOURCES

As databases proliferate, users want to access data from more than one source. For example, if several travel agents market their travel packages through the Web, customers would like to look at packages from different agents and compare them. A more traditional example is that large organizations typically have several databases, created (and maintained) by different divisions such as Sales, Production, and Purchasing. While these databases contain much common information, determining the exact relationship between tables in different databases can be a complicated problem. For example, prices in one database might be in dollars per dozen items, while prices in another database might be in dollars per item. The development of XML DTDs (see Section 22.3.3) offers the promise that such *semantic mismatches* can be avoided if all parties conform to a single standard DTD. However, there are many legacy databases and most domains still do not have agreed-upon DTDs; the problem of semantic mismatches will be frequently encountered for the foreseeable future.

Semantic mismatches can be resolved and hidden from users by defining relational views over the tables from the two databases. Defining a collection of views to give a group of users a uniform presentation of relevant data from multiple databases is called **semantic integration**. Creating views that mask semantic mismatches in a natural manner is a difficult task and has been widely studied. In practice, the task is made harder by the fact that the schemas of existing databases are often poorly

documented; thus, it is difficult to even understand the meaning of rows in existing tables, let alone define unifying views across several tables from different databases.

If the underlying databases are managed using different DBMSs, as is often the case, some kind of 'middleware' must be used to evaluate queries over the integrating views, retrieving data at query execution time by using protocols such as Open Database Connectivity (ODBC) to give each underlying database a uniform interface, as discussed in Chapter 5. Alternatively, the integrating views can be materialized and stored in a data warehouse, as discussed in Chapter 23. Queries can then be executed over the warehoused data without accessing the source DBMSs at run-time.

28.3 MOBILE DATABASES

The availability of portable computers and wireless communications has created a new breed of nomadic database users. At one level these users are simply accessing a database through a network, which is similar to distributed DBMSs. At another level the network as well as data and user characteristics now have several novel properties, which affect basic assumptions in many components of a DBMS, including the query engine, transaction manager, and recovery manager;

- Users are connected through a wireless link whose bandwidth is ten times less than Ethernet and 100 times less than ATM networks. Communication costs are therefore significantly higher in proportion to I/O and CPU costs.
- Users' locations are constantly changing, and mobile computers have a limited battery life. Therefore, the true communication costs reflect connection time and battery usage in addition to bytes transferred, and change constantly depending on location. Data is frequently replicated to minimize the cost of accessing it from different locations.
- As a user moves around, data could be accessed from multiple database servers within a single transaction. The likelihood of losing connections is also much greater than in a traditional network. Centralized transaction management may therefore be impractical, especially if some data is resident at the mobile computers. We may in fact have to give up on ACID transactions and develop alternative notions of consistency for user programs.

28.4 MAIN MEMORY DATABASES

The price of main memory is now low enough that we can buy enough main memory to hold the entire database for many applications; with 64-bit addressing, modern CPUs also have very large address spaces. Some commercial systems now have several *gigabytes* of main memory. This shift prompts a reexamination of some basic DBMS

design decisions, since disk accesses no longer dominate processing time for a memory-resident database:

- Main memory does not survive system crashes, and so we still have to implement logging and recovery to ensure transaction atomicity and durability. Log records must be written to stable storage at commit time, and this process could become a bottleneck. To minimize this problem, rather than commit each transaction as it completes, we can collect completed transactions and commit them in batches; this is called **group commit**. Recovery algorithms can also be optimized since pages rarely have to be written out to make room for other pages.
- The implementation of in-memory operations has to be optimized carefully since disk accesses are no longer the limiting factor for performance.
- A new criterion must be considered while optimizing queries, namely the amount of space required to execute a plan. It is important to minimize the space overhead because exceeding available physical memory would lead to swapping pages to disk (through the operating system's virtual memory mechanisms), greatly slowing down execution.
- Page-oriented data structures become less important (since pages are no longer the unit of data retrieval), and clustering is not important (since the cost of accessing any region of main memory is uniform).

28.5 MULTIMEDIA DATABASES

In an object-relational DBMS, users can define ADTs with appropriate methods, which is an improvement over an RDBMS. Nonetheless, supporting just ADTs falls short of what is required to deal with very large collections of **multimedia objects**, including audio, images, free text, text marked up in HTML or variants, sequence data, and videos. Illustrative applications include NASA's EOS project, which aims to create a repository of satellite imagery, the Human Genome project, which is creating databases of genetic information such as GenBank, and NSF/DARPA's Digital Libraries project, which aims to put entire libraries into database systems and then make them accessible through computer networks. Industrial applications such as collaborative development of engineering designs also require multimedia database management, and are being addressed by several vendors.

We outline some applications and challenges in this area:

- **Content-based retrieval:** Users must be able to specify selection conditions based on the contents of multimedia objects. For example, users may search for images using queries such as: "Find all images that are similar to this image" and "Find all images that contain at least three airplanes." As images are inserted into

the database, the DBMS must analyze them and automatically *extract features* that will help answer such content-based queries. This information can then be used to search for images that satisfy a given query, as discussed in Chapter 26. As another example, users would like to search for documents of interest using information retrieval techniques and keyword searches. Vendors are moving towards incorporating such techniques into DBMS products. It is still not clear how these domain-specific retrieval and search techniques can be combined effectively with traditional DBMS queries. Research into abstract data types and ORDBMS query processing has provided a starting point, but more work is needed.

- **Managing repositories of large objects:** Traditionally, DBMSs have concentrated on tables that contain a large number of tuples, each of which is relatively small. Once multimedia objects such as images, sound clips, and videos are stored in a database, individual objects of very large size have to be handled efficiently. For example, compression techniques must be carefully integrated into the DBMS environment. As another example, distributed DBMSs must develop techniques to efficiently retrieve such objects. Retrieval of multimedia objects in a distributed system has been addressed in limited contexts, such as client-server systems, but in general remains a difficult problem.

- **Video-on-demand:** Many companies want to provide video-on-demand services that enable users to dial into a server and request a particular video. The video must then be delivered to the user's computer in real time, reliably and inexpensively. Ideally, users must be able to perform familiar VCR functions such as fast-forward and reverse. From a database perspective, the server has to contend with specialized real-time constraints; video delivery rates must be synchronized at the server and at the client, taking into account the characteristics of the communication network.

28.6 GEOGRAPHIC INFORMATION SYSTEMS

Geographic Information Systems (GIS) contain spatial information about cities, states, countries, streets, highways, lakes, rivers, and other geographical features, and support applications to combine such spatial information with non-spatial data. As discussed in Chapter 26, spatial data is stored in either raster or vector formats. In addition, there is often a temporal dimension, as when we measure rainfall at several locations over time. An important issue with spatial data sets is how to integrate data from multiple sources, since each source may record data using a different coordinate system to identify locations.

Now let us consider how spatial data in a GIS is analyzed. Spatial information is most naturally thought of as being overlaid on maps. Typical queries include "What cities lie on I-94 between Madison and Chicago?" and "What is the shortest route from Madison to St. Louis?" These kinds of queries can be addressed using the techniques

discussed in Chapter 26. An emerging application is in-vehicle navigation aids. With Global Positioning Systems (GPS) technology, a car's location can be pinpointed, and by accessing a database of local maps, a driver can receive directions from his or her current location to a desired destination; this application also involves mobile database access!

In addition, many applications involve interpolating measurements at certain locations across an entire region to obtain a *model*, and combining overlapping models. For example, if we have measured rainfall at certain locations, we can use the TIN approach to triangulate the region with the locations at which we have measurements being the vertices of the triangles. Then, we use some form of interpolation to estimate the rainfall at points within triangles. Interpolation, triangulation, map overlays, visualizations of spatial data, and many other domain-specific operations are supported in GIS products such as ESRI Systems' ARC-Info. Thus, while spatial query processing techniques as discussed in Chapter 26 are an important part of a GIS product, considerable additional functionality must be incorporated as well. How best to extend ORDBMS systems with this additional functionality is an important problem yet to be resolved. Agreeing upon standards for data representation formats and coordinate systems is another major challenge facing the field.

28.7 TEMPORAL AND SEQUENCE DATABASES

Currently available DBMSs provide little support for queries over ordered collections of records, or *sequences*, and over temporal data. Typical sequence queries include "Find the weekly moving average of the Dow Jones Industrial Average," and "Find the first five consecutively increasing temperature readings" (from a trace of temperature observations). Such queries can be easily expressed and often efficiently executed by systems that support query languages designed for sequences. Some commercial SQL systems now support such SQL extensions.

The first example is also a temporal query. However, temporal queries involve more than just record ordering. For example, consider the following query: "Find the longest interval in which the same person managed two different departments." If the period during which a given person managed a department is indicated by two fields *from* and *to*, we have to reason about a collection of intervals, rather than a sequence of records. Further, temporal queries require the DBMS to be aware of the anomalies associated with calendars (such as leap years). Temporal extensions are likely to be incorporated in future versions of the SQL standard.

A distinct and important class of sequence data consists of DNA sequences, which are being generated at a rapid pace by the biological community. These are in fact closer to sequences of characters in text than to time sequences as in the above examples. The field of biological information management and analysis has become very popular

in recent years, and is called **bioinformatics**. Biological data, such as DNA sequence data, is characterized by complex structure and numerous relationships among data elements, many overlapping and incomplete or erroneous data fragments (because experimentally collected data from several groups, often working on related problems, is stored in the databases), a need to frequently change the database *schema* itself as new kinds of relationships in the data are discovered, and the need to maintain several versions of data for archival and reference.

28.8 INFORMATION VISUALIZATION

As computers become faster and main memory becomes cheaper, it becomes increasingly feasible to create visual presentations of data, rather than just text-based reports. Data visualization makes it easier for users to understand the information in large complex datasets. The challenge here is to make it easy for users to develop visual presentation of their data and to interactively query such presentations. Although a number of data visualization tools are available, efficient visualization of large datasets presents many challenges.

The need for visualization is especially important in the context of decision support; when confronted with large quantities of high-dimensional data and various kinds of data summaries produced by using analysis tools such as SQL, OLAP, and data mining algorithms, the information can be overwhelming. Visualizing the data, together with the generated summaries, can be a powerful way to sift through this information and spot interesting trends or patterns. The human eye, after all, is very good at finding patterns. A good framework for data mining must combine analytic tools to process data, and bring out latent anomalies or trends, with a visualization environment in which a user can notice these patterns and interactively drill down to the original data for further analysis.

28.9 SUMMARY

The database area continues to grow vigorously, both in terms of technology and in terms of applications. The fundamental reason for this growth is that the amount of information stored and processed using computers is growing rapidly. Regardless of the nature of the data and its intended applications, users need database management systems and their services (concurrent access, crash recovery, easy and efficient querying, etc.) as the volume of data increases. As the range of applications is broadened, however, some shortcomings of current DBMSs become serious limitations. These problems are being actively studied in the database research community.

The coverage in this book provides a good introduction, but is not intended to cover all aspects of database systems. Ample material is available for further study, as this

chapter illustrates, and we hope that the reader is motivated to pursue the leads in the bibliography. Bon voyage!

BIBLIOGRAPHIC NOTES

[288] contains a comprehensive treatment of all aspects of transaction processing. An introductory textbook treatment can be found in [77]. See [204] for several papers that describe new transaction models for nontraditional applications such as CAD/CAM. [1, 668, 502, 607, 622] are some of the many papers on real-time databases.

Determining which entities are the same across different databases is a difficult problem; it is an example of a semantic mismatch. Resolving such mismatches has been addressed in many papers, including [362, 412, 558, 576]. [329] is an overview of theoretical work in this area. Also see the bibliographic notes for Chapter 21 for references to related work on multidatabases, and see the notes for Chapter 2 for references to work on view integration.

[260] is an early paper on main memory databases. [345, 89] describe the Dali main memory storage manager. [359] surveys visualization idioms designed for large databases, and [291] discusses visualization for data mining.

Visualization systems for databases include DataSpace [515], DEVise [424], IVEE [23], the Mineset suite from SGI, Tioga [27], and VisDB [358]. In addition, a number of general tools are available for data visualization.

Querying text repositories has been studied extensively in information retrieval; see [545] for a recent survey. This topic has generated considerable interest in the database community recently because of the widespread use of the Web, which contains many text sources. In particular, HTML documents have some structure if we interpret links as edges in a graph. Such documents are examples of semistructured data; see [2] for a good overview. Recent papers on queries over the Web include [2, 384, 457, 493].

See [501] for a survey of multimedia issues in database management. There has been much recent interest in database issues in a mobile computing environment, for example, [327, 337]. See [334] for a collection of articles on this subject. [639] contains several articles that cover all aspects of temporal databases. The use of constraints in databases has been actively investigated in recent years; [356] is a good overview. Geographic Information Systems have also been studied extensively; [511] describes the Paradise system, which is notable for its scalability.

The book [695] contains detailed discussions of temporal databases (including the TSQL2 language, which is influencing the SQL standard), spatial and multimedia databases, and uncertainty in databases. Another SQL extension to query sequence data, called SRQL, is proposed in [532].