A rnore radical file organization is to regard the fact table as a large Illuitidi-rnensional array and store it and index it as such. This approach is taken in MOLAP systerns. Since the array is lIluch larger than available lnain lnelnory, it is broken up into contiguous chunks, as discussed in Section 23.8. In addition, traditional B+- tree indexes are created to enable quick retrieval of chunks that contain tuples with values in a given range for one or rnore dilnensions.

25.7 DATA WAREHOUSING

Data warehouses contain consolidated data from many sources, augrnented with sunnnary infornation and covering a long time period. Warehouses are lnuch larger than other kinds of databases; sizes ranging frorn several gigabytes to terabytes are cornman. Typical workloads involve ad hoc, fairly cOlllplex queries and fast response times are important. These characteristics differentiate warehouse applications from OL'TP applications, and different DBMS design and irnplerrlentation techniques nUlst be used to achieve satisfactory results. A distributed DBMS with good scalability and high availability (achieved by storing tables redundantly at more than one site) is required for very large warehouses.

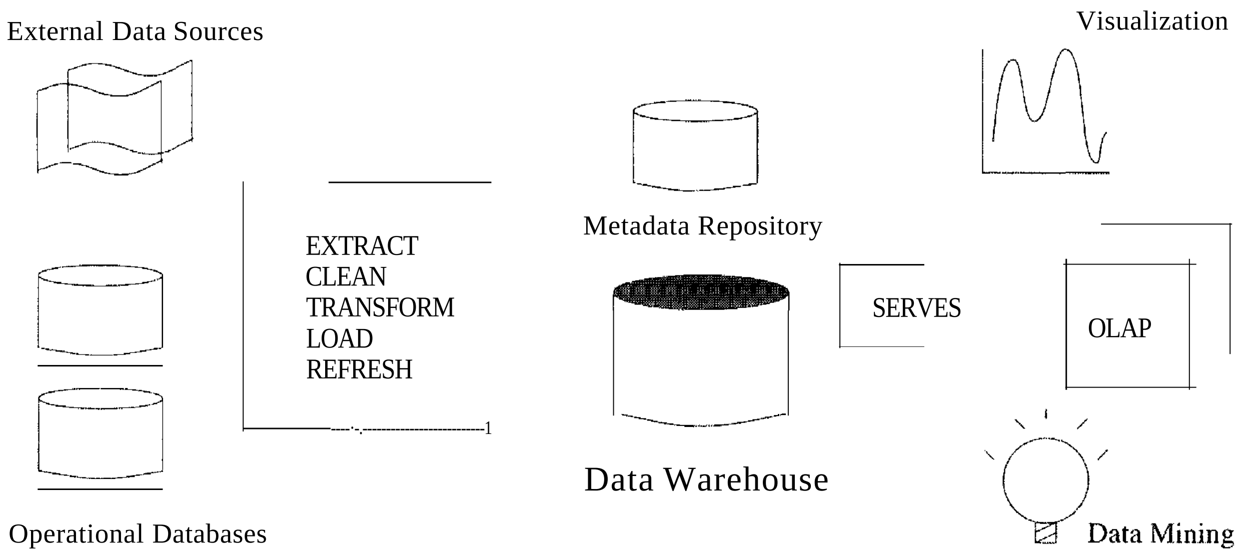

Figure 25.10 A rrypical Data \Varehousing Architecture

A typical data warehousing architecture is illustrated in Figure 25.10. An organization's daily operations access and rnodify operational databases. Data frorl1 these operational databases and other external sources (e.g., custorner profiles supplied by external consultants) are extracted by using interfaces such as JI)BC (see Section 6.2).

25.7.1 Creating and Maintaining a Warehouse

Many challenges rnust be Inet in creating and Inaintaining a large data warehouse.A good database scherua nlust be designed to hold an integrated collection of data copied froIn diverse sources. For exarnple, a cornpany warehouse rnight include the inventory and personnel departrnents' databases, together with sales databases rnaintained by offices in different countries. Since the source databases are often created and rnaintained by different groups, there are a nUlnber of selnantic Inisrnatches across these databases, such as different currency units, different narnes for the saIne attribute, and differences in how tables are nornlalized or structured; these differences Inust be reconciled when data is brought into the warehouse. After the warehouse schenla is designed, the warehouse must be populated, and over tirne, it Inust be kept consistent with the source databases.

Data is extracted from operational databases and external sources, cleaned to Inininlize errors and fill in Inissing information when possible, and **trans**formed to reconcile semantic Inismatches. Transforlning data is typically accOlnplished by defining a relational view over the tables in the data sources (the operational databases and other external sources). Loading data consists of ruaterializing such views and storing therll in the warehouse. Unlike a standard view in a relational DBMS, therefore, the view is stored in a database (the warehouse) that is different frorn the database(s) containing the tables it is defined over.

The cleaned and transfonned data is finally loaded into the warehouse. Additional preprocessing such as sorting and generation of surnrnary information is carried out at this stage. Data is partitioned and indexes are built for efficiency. Due to the large vohllue of elata, loading is a slow process. Loading a terabyte of data sequentially can take weeks, and loading even a gigabyte can take hours. Parallelisul is therefore iInportant for loading warehouses.

AJter data is loaded into a warehouse, additional rneasures rnust be taken to ensure that the data in the warehouse is periodically refreshed to reflect updates to the data sources and periodically purge old data (perhaps onto archival rnedia). Observe the connection between the problern of refreshing warehouse tables and a,synchronously rnaintaining replicas of tables in a distributed DBMS. Maintaining replicas of source relations is an essential part of warehousing, and this application clornain is an iInportant factor in the popularity of asynchronous replication (Section 22.11.2), even though asynchronous replication violates the principle of distributed data independence. The problern of refreshing warehouse tables (\vhich are rnaterialized views over tables in the source databases) has also renewed interest in inerernental Illaintenance of materialized views. (We discuss rnaterialized views in Section 25.8.)

An irnportant task in maintaining a warehouse is keeping track of the data currently stored in it; this bookkeeping is done by storing infofrnation about the warehouse data in the systenl catalogs. The systerIl catalogs associated with a \varehouse are very large and often stored and 111anaged in a separate database called a **metadata repository**. The size and cornplexity of the catalogs is in part due to the size and cOlnplexity of the warehouse itself and in part because a lot of adrninistrative inforrnation rnust be Inaintained. For example, we HlllSt keep track of the source of each warehouse table and when it was last refreshed, in addition to describing its fields.

1'he value of a warehouse is ultin1ately in the analysis it enables. The data in a warehouse is typically accessed and analyzed using a variety of tools, including OLAP query engines, data. mining algorithrns, inforrnation visualization tools, statistical packages, and report generators.

25.8 VIEWS AND DECISION SUPPORT

Views are widely used in decision support applications. Different groups of analysts within an organization are typically concerned with different aspects of the business, and it is convenient to define views that give each group insight into the business details that concern it. Once a view is defined, we can write queries or new view definitions that use it, as we saw in Section 3.6; in this respect a view is just like a base table. Evaluating queries posed against views is very ilnportant for decision support applications. In this section, we consider how such queries can be evaluated efficiently after placing views within the context of decision support applications.

25.8.1 Views, OLAP, and Warehousing

Views are closely related to OLAP and data warehousing.

OLAP queries are typically aggregate queries. Analysts want fast answers to these queries over very large datasets, and it is natural to consider precoluputing views (see SectiorlS 25.9 and 25.10). In particular, the CUBE operator- -discussed in Section 25.3—gives rise to several aggregate queries that are closely related. The relationships that exist between the Inany aggregate queries that arise froln a single CUBE operation can be exploited to develop very effective precornpu-tation strategies. The idea is to choose a subset of the aggregate queries for Inaterialization in such a way that typical CUBE queries can be quickly answered by using the materialized views arld doing S(Hne additional cornplltation. The choice of views to 111aterialize is influenced by how lllany queries they can potentially speed up and by the aillount of space required to store the Inaterialized view (since we have to work with a given alnount of storage space).

A data \varehouse is just a collection of asynchronously replicated tables and periodically synchronized views. A warehouse is characterized by its size, the nuruber of tables involved, and the fact that IllOSt of the underlying tables are froln external, independently lnaintained databases. Nonetheless, the fundaluental probleln in warehouse lnaintenance is asynchronous rnaintenance of replicated tables and materialized views (see Section 25.10).

25.8.2 Queries over Views

Consider the following view, RegionalSales, which cornputes sales of products by category and state:

```
CREATE VIEW RegionalSales (category, sales, state)
        AS SELECT P.category, S.sales, L.state
           FROM   Products P, Sales S, Locations L
           WHERE  P.pid = S.pid AND S.locid = L.locid
```

The following query computes the total sales for each category by state:

```
SELECT    R.category, R.state, SUM (R.sales)
FROM      RegionalSales R
GROUP BY  R.category, R,.state
```

While the SQL standard does not specify how to evaluate queries on views, it is useful to think in ternlS of a process called **query modification**. rrhe idea is to replace the occurrence of RegionalSales in the query by the view definition. The result on this query is

```
SELECT    H,.category, R.state, SUM (R.sales)
FROM      ( SELECT P.category, S.sales, L.state
            FROM   Products P, Sales S, Locations L
            WHERE  P.piel = S.pid AND S.locid = L.locid ) AS R
GROUP BY  R.category, R.state
```

25.9 VIEW MATERIALIZATION

We can answer a query on a view by using the query rnodification technique just described. Often, however, queries against cornplex view definitions Illust be answered very fast because users engaged in decision support activities require interactive response tirIles. Even with sophisticated optimization and evaluation techniques, there is a lirnit to how fast we can answer such queries. Also, if the underlying tables are in a rernote database, the query rIlodification approach rnay not even be feasible because of issues like connectivity and availability.

An alternative to query rnodification is to precornpute the view definition and store the result. When a query is posed on the view, the (unrllodified) query is executed directly on the precornputed result. This approach, called **view materialization**, is likely to be rnuch faster than the query modification approach because the complex view need not be evaluated when the query is computed. Materialized views can be used during query processing in the sarne way as regular relations; for exarnple, we can create indexes on nlaterialized views to further speed up query processing. The drawback, of course, is that we must maintain the consistency of the precomputed (or *m,aterialized)* view whenever the underlying tables are updated.

25.9.1 Issues in View Materialization

Three questions must be considered with regard to view nlaterialization:

1. What views should we rnaterialize and what indexes should we build on the rnaterialized views?
2. Given a query on a view and a set of materialized views, can we exploit the rnaterialized views to answer the query?
3. I-Iow should we synchronize rnaterialized views with changes to the underlying tables? The choice of synchronization technique depends on several factors, such as whether the underlying tables are in a rernote database. We discuss this issue in Section 25.10.

'rhe answers to the first two questions are related. The choice of views to rnaterialize and index is governed by the expected workload, and the discussion of indexing in Chapter 20 is relevant to this question as well. The choice of views to rnaterialize is rnore cornplex than just choosing indexes on a set of database tables, however, because the range of alternative views to rnaterialize is wider. The goal is to rnaterialize a srnaU, carefully chosen set of views that can be utilized to quickly answer rnost of the irnportant queries. COIlversely, once we have chosen a set of views to rnaterialize, we have to consider how they can be used to answer a, given query.

Consider the RegionalSales view. It involves a JOIn of Sales, Products, and Locations and is likely to be expensive to cornpute. On the other hand, if it is rnaterialized and stored with a clustered B+ tree index on the search key (category, state, sales), we Gall ans\ver the exarnple query by an index-only searl.

Given the rnaterialized view and this index, we can also answer queries of the follo\ving forrn efficiently:

```
SELECT     R.state, SUM (R.sales)
FROM       RegionalSales R
WHERE      R.category == 'Laptop'
GROUP BY   R.state
```

To answer such a query, we can use the index on the Inaterialized view to locate the first index leaf entry with *category* == 'Laptop' and then scan the leaf level until we come to the first entry with *category* not equal to Laptop.

The given index is less effective on the following query, for which we are forced to scan the entire leaf level:

```
SELECT     R.state, SUM (R.sales)
FROM       R,egionalSales R
WHERE      R.state == 'Wisconsin'
GROUP BY   R.category
```

This exanlple indicates how the choice of views to materialize and the indexes to create are affected by the expected workload. This point is illustrated further by our next exarnple.

Consider the following two queries:

```
SELECT     P.category, SUM (S.sales)
FROM       Products P, Sales S
WHERE      P.pic! == S.pic!
GROUP BY   P.category

SELECT     L.state, SUM (S.sales)
FROM       Locations I;, Sa1es S
WHERE      .L.locid = S.locid
GROUP BY   L.state
```

These two queries require us to join the SaJes table (which is likely to be very large) with another table and aggregate the result. How can we use rnaterialization to speed up these queries? The straightforward approach is to precornpute each of the joins involved (Products with Sales and Locations with Sales) or to preconlpute each query in its entirety. An alternative approach is to define the following view:

```
CREATE   VIEW rrotalSaJes (pid, locid, total)
         AS SELECT   S.pid, S.locid, SUM (S.sales)
            FROM     Sales S
            GROUP BY S.pid, S.locid
```

The view TotalSales can be rnaterialized and used instead of Sales in our two exalnple queries:

```
SELECT   P.category, SUM (T.total)
FROM     Products P, TotalSales T
WHERE    P.pid = T.pid
GROUP BY P.category

SELECT   L.state, SUM (T.total)
FROM     Locations L, TotalSales T
WHERE    L.locid = rr.locid
GROUP BY L.state
```

25.10 MAINTAINING MATERIALIZED VIEWS

A materialized view is said to be **refreshed** when we rnake it consistent with changes to its underlying tables. rrhe process of refreshing a view to keep it consistent with changes to the underlying table is often referred to as **view maintenance**. Two questions to consider are

1. *flow* do we refresh a view' when an underlying table is nlodified? Two issues of particular interest are how to Inaintain views *incrementally*, that is, without recornputing froll! scratch when there is a change to an underlying table; and how to rnaintain views in a distributed environrnent such as a data warehouse.

2. *When* should we refresh a view in response to a change to an underlying table?

25.10.1 Incremental View Maintenance

A straightforward approach to refreshing a view is to sirnply recompute the view when an underlying table is rnodified. This rnay, in fact, be a reasonable strategy in sorne cases. For exarnple, if the underlying tables are in a rernote database, the view can be periodically recornputed and sent to the data warehouse \vhere the vie\v is Hlaterialized. This has the advantage that the underlying tables need not be replicated at the warehouse.

\Vhenever possible, however, algorithrns for refreshing a view should be incre**mental**, in that the cost is proportional to the extent of the change rather than the cost of recornputing the view fr(Hn scratch.

To understand the intuition behind incrernental view rnaintenance algorithnls, observe that a given row in the rnaterialized view can appear several times, depending on how often it was derived. (R.ecall that duplicates are not elirninated fro111 the result of an SQL query unless the DISTINCT clause is used. In this section, we discuss rnultiset sernantics, even when relational algebra notation is used.) The rHain idea behind incremental rnaintenance algorithrIls is to efficiently compute changes to the rows of the view, either new rows or changes to the count associated with a row; if the count of a row becornes 0, the row is deleted frorH the view.

We present an incrernental maintenance algorithnl for views defined using projection, binary join, and aggregation; we cover these operations because they illustrate the rHain ideas. The approach can be extended to other operations such as selection, un.ion, intersection, and (rnultiset) difference, as well as expressions containing several operators. The key idea is still to rnaintain the nurnber of derivations for each view row, but the details of how to efficiently conlpute the changes in view rows and associated counts differ.

Projection Views

Consider a view V defined in tenns of a projection on a tableR; that is, V = $n(R)$. Every row v in V has an associated count, corresponding to the nurnber of tirnes it can be derived, which is the nurnber of rows in R that yield v when the projection is applied. Suppose we 1nodifyR by inserting a collection of rows R_i and deleting a collection of existing 1'o\vs R_d.[1] We cornpute $\pi(R_i)$ and add it to V. If the multiset $\pi(R_i)$ contains a row r with count c and r does not appear in V, we add it to V"with count c. If r is in V, we add c to its count. We also cornpute $n(Rd)$ and subtract it fronl V. ()bserve that if r appears in $\pi(R_d)$ with count c, it 111USt also appear in V with a higher count;2 we subtract c frOTH r's count in V".

1'These collections can be multisets of rows. We can treat a row rnodification as an insert followed by a delete, for sirnplicity.

2As a simple exercise, consider why this rnust be so.

As an exanlple, consider the view $\pi_{sales}(Sales)$ and the instance of Sales shown in Figure 25.2. _Each row in the vie\v has a single cohunn; the (n)\vwith) value 25 appears with count 1, and the value 10 appears with count 3. If we delete one of the rows in Sales \vith *sales* 10, the count of the (row with) value 10 in the view becornes 2. If we insert a new row into Sales with *sales* 99, the view no\v has a row with value 99.

An hnportant point is that we have to rnaintain the counts associated with rows even if the view definition uses the DISTINCT clause, rneaning that duplicates are elilninated frorn the view. Consider the salne view with set semantics—the DISTINCT clause is used in the SQL view definition··------and suppose that we delete one of the rows in Sales with *sales* 10. Does the view now contain a row with value 10'1 To deterrIline that the answer is yes, we need to maintain the row counts, even though each row (with a nonzero count) is displayed only once in the Inaterialized view.

Join Views

Next, consider a view V defined as a join of two tables, R [x] S. Suppose we modify R by inserting a collection of rows R_i and deleting a collection of rows Rd. We cornpute Ri [x] S and add the result to V. We also C0111pute $Rd \bowtie S$ and subtract the result frorl1 V. Observe that if r appears in R_d [x] S with count c, it rnust also appear in V with a higher count.[3]

Views with Aggregation

Consider a view V defined over R using GROUP BY on colUllln G and an aggregate operation on colu1nn A. Each row v in the view surnrnarizes a group of tuples in R and is of the fonn $(g,\ summary)$, where 9 is the value of the grouping colulnn G and the sununary inforInation depends on the aggregate operation. To lnaintain such a view incrernentally, in general, we have to keep a lnore detailed surrllnary than just the inforrnation included in the view. If the aggregate operation is COUNT, we need to Inaintain only a count c for each row v in the view. If a ro\v r is inserted intoR, and there is no row v in V with '$v.G = r.G$, we add a new row $(r.G,$ 1). If there is a ro,v v \vith $v.C\} = r.G$, we incrernent its count. If a row r is deleted fro111 R, we decrcrnent the count for the row v with $v.Ci = r.G$; v can be deleted if its count becornes 0, because then the last row in this group has been deleted frorn R.

If the aggregate operation is SUM, we have to lllaintain a suiii s and also a count c. If a row r is inserted into R and there is no row v in V with $v.C; = T.C:,$

[3] As another simple exercise, consider why this mllst be so.

we add a new row $\langle r.G, a, 1)$. If there is a row $\langle r.G, s, c\rangle$, we replace it by (*r.Ci, s* + a, c + 1). If a row r is deleted frolll R, \ve replace the row $\langle r.G, s$, c) with $\langle r.G, s$ - *a, c* - 1); v can be deleted if its count becornes 0. Observe that without the count, we do not know when to delete *'u,* since the Slun for a group could be 0 even if the group contains SCHne rows.

If the aggregate operation is AVG, we have to lllaintain a Slun s, a count c, and the average for each row in the view. The SlUll and count are rnaintained incrernentally as already described, and the average is corllputed as *s*/ c.

The aggregate operations MIN and MAX are potentially expensive to rnaintain. Consider MIN. For each group in *R,* we rnaintain *(g, rn, c),* where *rn* is the minimum value for colUllln *A* in the group *g,* and c is the count of the nUlllber of rows *r* in *R* with *r.G* = *9* and *r.A* = m. If a row *r* is inserted into Rand *r.G* = *g,* if *r.A* is greater than the miniriulill m for group *g,* we can ignore *r*. If *r.A* is equal to the 111iniInurll m for *r's* group, we replace the summary row for the group with *(g,* m, c+ 1). If *r.A* is less than the minirllum m for r's group, we replace the SUlnrnary for the group with *(g, T.A,* 1). If a row *r* is deleted frorn Rand *T.A* is equal to the minimurIl *m* for T'S group, then we HUlst decrernent the count for the group. If the count is greater than 0, we sinlply replace the surnmary for the group with *(g, rn, c-*_· 1). However, if the count becomes 0, this Ineans the last row with the recorded rninimum *A* value has been deleted from *R* and we have to retrieve the sInallest *A* value among the relnaining rows in *R* with- group value r.G-and this might require retrieval of all rows in *R* with group value *r.G*.

25.10.2 Maintaining Warehouse Views

The views rnaterialized in a data warehouse can be based on source tables in rernote databases. rIhe asynchronous replication techniques discussed in Section 22.11.2 allow us to connnunicate changes at the source to the warehouse, but refreshing views incrernentally in a distributed setting presents sorne unique challenges. To illustrate this, we consider a sirnpleview that identifies suppliers of Toys.

```
CREATE VIEW ToySuppliers (sid)
       AS SELECT S.sid
          FROM   Suppliers S, Products P
          WHERE  S.pid = P.piel AND P.category = 'Toys'
```

Suppliers is a new table introduced for this exarnple; let us assume that it has just two fields, *sid* aIId *pid*, indicating that supplier *sid* supplies part *pid*. The location of the tables Proclucts and Suppliers and the view ToySuppliers influences how we IIlaintain the view. Suppose that all three are rnaintained at a single site. We can Inaintain the view increlnentally using the techniques discussed in Section 25.10.1. If a replica of the vie\v is created at another site, we can 1l1onitor changes to the Inaterialized vie\v and apply thcIn at the second site using the asynchronous replication techniques froIn Section 22.11.2.

But, what if Products and Suppliers are at one site and the view is Inaterialized (only) at a second site? To rnotivate this scenario, we observe that, if the first site is used for operational data and the second site supports cornplex analysis, the two sites lnay well be adrninistered by different groups. The option of lnaterializing ToySuppliers (a view of interest to the second group) at the first site (run by a different group) is not attractive and may not even be possible; the adnlinistrators of the first site may not want to deal with someone else's views, and the a(hninistrators of the second site n1ay not want to coordinate with sonleone else whenever they Inodify view definitions. As another motivation for rnaterializing views at a different location froIn source tables, observe that Products and Suppliers may be at two different sites. Even if ·we rnaterialize ToySuppliers at one of these sites, one of the two source tables is reillote.

Now that we have presented Inotivation for rnaintaining rroySuppliers at a location (say, Warehouse) different froIn the one (say, Source) that contains Products and Suppliers, let us consider the difficulties posed by data distribution. Suppose that a new Products record (with *category* = 'Toys') is inserted. We could try to rnaintain the view incren1entally as follows:

1. The Warehouse site sends this update to the Source site.
2. To refresh the view, we need to check the Suppliers table to find suppliers of the itern, and so the v\larehouse site asks the Source site for this inforrnation.
3. The Source site returns the set of suppliers for the sold iteln, and the Warehouse site incrernentally refreshes the view.

This works when there are no additional changes at the Source site in between steps (1) and (3). If there are changes, however, the Inaterializecl view can becorne incorrect   reflecting a state that can never arise except for anornalies introduced by the preceding, naive, increlnental refresh algorithrn. To see this, suppose that Products is enlpty and Suppliers contains just the row $\langle s1, 5\rangle$ initially, and consider the following sequence of events:

1. Product *pid* = 5 is inserted \vith *category* = 'Toys'; Source notifies\Varehouse.
2. Warehouse asks Source for suppliers of product *pid* = 5. *(The only such supplier at this instant is 81.)*

3. The row (82,5) is inserted into Suppliers; Source notifies \Varehouse.
4. To decide whether 82 should be added to the view, we need to know the category of product *pid* = 5, and \Varehouse asks Source. *(Warehouse has not received an answer to its previous question.)*
5. Source now processes the first query frorn Warehouse, finds two suppliers for part 5, and returns this inforrnation to Warehouse.
6. Warehouse gets the answer to its first question: suppliers 81 and 82, and adds these to the view, each with count 1.
7. Source processes the second query frorn \Varehouse and responds with the inforlll1ation that part 5 is a toy.
8. Warehouse gets the answer to its second question and accordingly incre-Hlents the count for supplier 82 in the view.
9. Product *pid* = 5 is now deleted; Source notifies Warehouse.
10. Since the deleted part is a toy, Warehouse decrements the counts of nlatching view tuples; 81 has count 0 and is relnoved, but *s2* has count 1 and is retained.

Clearly, 82 should not rernain in the view after part 5 is deleted. This example illustrates the added subtleties of incremental view rnaintenance in a distributed environment, and this is a topic of ongoing research.

25.10.3 When Should We Synchronize Views?

A **view maintenance policy** is a decision about when a view is refreshed, independent of whether the refresh is incrernental or not. A view can be refreshed within the sallIe transaction that updates the underlying tables. This is called **immediate view Iuaintenance**. The update transaction is slowed by the refresh step, and the irupact of refresh increases with the nurnber of materialized views that depend on the updated table.

Alternatively, we can defer refreshing the view. Updates are captured in a log and applied subsequently to the rnaterialized vic\vs. There are several **deferred view maintenance** policies:

1. **Lazy:** The rnaterialized view V is refreshed at the tilne a query is evaluated using V, if V is not already consistent with its underlying base tables. This approach slows down queries rather than updates, in contrast to iHnnediate view rnaintenance.

Views for De-cision **Support:** DBMS vendors are enhancing their main relational products to support decision support queries. IBM DB2 supports materialized views with transaction-consistent or user-invoked maintenance. Microsoft SQL Server supports partition views, \vhich are unions of (ruany) horizontal partitions of a table. These aJ'e airned at a warehollsing envirOllrnent where each partition could be, for exalnple, a rnonthly update. Queries on partition vie\vs are opthnized so that only relevant partitions are accessed. Oracle 9i supports 111aterialized views with transaction-consistent, user-invoked, or tilne-scheduled nlaintenance.

2. Periodic: The lllaterialized view is refreshed periodically, say, once a day. The discussion of the Capture and Apply steps in asynchronous replication (see Section 22.11.2) should be reviewed at this point, since it is very relevant to periodic view lllaintenance. In fact, many vendors are extending their asynchronous replication features to support lllaterialized views. Materialized views that are refreshed periodically are also called snapshots.
3. Forced: rrhe rnaterialized view is refreshed after a certain nurnber of changes have been made to the underlying tables.

In periodic and forced view nlaintenance, queries rllay see an instance of the IIlaterialized view that is not consistent with the current state of the underlying tables. That is, the queries would see a different set of rows if the view definition was recornputed. This is the price paid for fast updates and queries, and the trade-off is sirnilar to the trade-off rnade in using asynchronous replication.

25.11 REVIEW **QUESTIONS**

Answers to the review questions can be found in the listed sections.

- What are *decision support* applications? :Oiscuss the relationship of *complex SQL queries, OLA.P, data rnining,* and *data warehousing.* (Section 25.1)
- Describe the rnultidirnensional data luodel. Explain the distinction between *rneasurcs* and *dirnensions* and between *fact tables* and *dimension tables.* What is a *star 8chenz,a?* (Sections 25.2 and 25.2.1)
- Cornrnon OLAP operations have received special naInes: *roll-up, drill-down, pivoting, slicing,* and *dicing.* Describe each of these operations and illustrate thern using exarnples. (Section 25.3)
- I)escribe the SQL:1999 ROLLUP and CUBE features and their relationship to the ()LAP operations. (Section 25.3.1)

- Describe the SQL:1999 WINDOW feature, in particular, frarning and ordering of windows. How does it support queries over ordered data? Give exarnples of queries that are hard to express without this feature. (Section 25.4)
- New query paradigrns include *top N queries* and *online aggTegation.* Explain the nlotivation behind these concepts and illustrate then1 through exaruples. (Section 25.5)
- Index structures that are especially suitable for OLAP systen1s include *bitrnap indexes* and *join indexes.* Describe these structures. How are bitrnap indexes related to B+ trees? (Section 25.6)
- Information about daily operations of an organization is stored in *opeTational databases.* Why is a *data warehouse* used to store data frolH operational databases? What issues arise in data warehousing? Discuss *data extTaction, cleaning, transjoTrnation,* and *loading.* Discuss the challenges in efficiently *TejTeshing* and *purging* data. (Section 25.7)
- Why are views irnportant in decision support environments? How are views related to data warehousing and OLAP? Explain the *queTy modification* technique for answering queries over views and discuss why this is not adequate in decision support environrnents. (Section 25.8)
- What are the rnain issues to consider in maintaining materialized views? Discuss how to select views to materialize and how to use rnaterialized views to answer a query. (Section 25.9)
- How can views be rnaintained *incTernentally?* Discuss all the relational algebra operators and aggregation. (Section 25.10.1)
- Use an exarnple to illustrate the added cornplications for incrernental view maintenance introduced by data distribution. (Section 25.10.2)
- Discuss the choice of an appropriate *rnaintenance policy* for when to refresh a view. (Section 25.10.3)

EXERCISES

Exercise 25.1 Briefly answer the following questions:

1. How do warehousing, OLAP, and data rnining cornplernent each other?
2. What is the relationship between data warehousing and data replication? Which fornl of replication (synchronous or asynchronous) is better suited for data warehousing? Why?
3. What is the role of the rnetadata repository in a data warehouse'? How does it differ frorn a catalog in a relational DBMS?
4. What considenttions are involved in designing a data warehouse'?

5. Once a warehouse is designed and loaded, how is it kept current with respect to changes to the source databases?
6. One of the advantages of a warehouse is that we can use it to track how the contents of a relation change over titue; in contrast, we have only the current snapshot of a relation in a regular DBMS. Discuss how you would maintain the history of a relation *R*, taking into account that 'old' infonnation lllust sOlnehow be purged to rnake space for Hew infonnatioll.
7. Describe dilnensions and rneasures in the multidirnensional data model.
8. What is a fact table, and why is it so irnportant frOIn a performance standpoint?
9. What is the fundarnental difference between MOLAP and ROLAP systems?
10. \What is a star scheIna? Is it typicaU:y in BCNF? Why or why not?
11. How is data rnining different from OLAP?

Exercise 25.2 Consider the instance of the Sales relation shown in Figure 25.2.

1. Show the result of pivoting the relation on *pid* and *tirneid.*
2. Write a collection of SQL queries to obtain the same result as in the previous part.
3. Show the result of pivoting the relation on *pid* and *lacid.*

Exercise 25.3 Consider the cross-tabulation of the Sales relation shown in Figure 25.5.

1. Show the result of roll-up on *lacid* (i.e., state).
2. Write a collection of SQL queries to obtain the same result as in the previous part.
3. Show the result of roll-up on *lacid* followed by drill-down on *pid.*
4. Write a collection of SQL queries to obtain the same result as In the previous part, starting with the cross-tabulation shown in Figure 25.5.

Exercise 25.4 Briefly answer the following questions:

1. What is the differences between the WINDOW clause and the GROUP BY clause'?
2. Give an example query that cannot be expressed in SQL without the WINDOW clause but that can be expressed with the WINDOW clause.
3. What is the *frame* of a window in SQL:19997
4. Consider the fonowing simple GROUP BY query.

```
SELECT     T.year, SUM (S.sales)
FROM       Sales 5, Times T
WHERE      S.tilneid='T.timeid
GROUP BY   T.year
```

   Can you write this query in SQL:1999 without using a GROUP BY cIa.use? (Hint: Use the SQL:1999 WINDOW clause.)

Exercise 25.5 Consider the Locations, Products, and Sales relations shown in Figure 25.2. Write the following queries in SQL:1999 llsing the WINDOW clause whenever you need it.

1. Find the percentage change in the total IJ10nthly sales for each location.
2. Find the percentage change in the total quarterly sales for each product.

3. Find the average daily sales over the preceding 30 days for each product.
4. For each week, find the maximum uloving average of sales over the preceding four \veeks.
5. Find the top three locations ranked by total sales.
6. F'ind the top three locations ranked by curnulative sales, for every month over the past year.
7. Rank all locations by total sales over the past year, and for each location print the difference in total sales relative to the location behind it.

Exercise 25.6 Consider the CustOIuers relation and the bitmap indexes shown in Figure 25.9.

1. For the same data, if the underlying set of rating values is assuIued to range froIlI 1 to 10, show how the bitnlap indexes would change.
2. How would you use the bitIllap indexes to answer the following queries? **If** the bitmap indexes are not useful, explain why.
   (a) How many customers with a rating less than 3 are male?
   (b) What percentage of custoIners are male?
   (c) How rnany customers are there?
   (d) How many custonlers are named Woo?
   (e) Find the rating value with the greatest number of customers and also find the nUIll-bel' of custorners with that rating value; if several rating values have the maxirnurn number of custoIllers, list the requested infonuation for all of theIn. (AssuIne that very few rating values have the same nUluber of customers.)

Exercise **25.7** In addition to the Customers table of Figure 25.9 with bitrnap indexes on *gender* and *'rating,* assurne that you have a table called Prospects, with fields *rating* and *prospectid.* This table is used to identify potential customers.

1. Suppose that you also have a bitrnap index on the *rating* field of Prospects. Discuss whether or not the bitnlap indexes would help in corllputing the join of Custorners and Prospects on *rating.*
2. Suppose you have *no* bitrnap index on the *rating* field of Prospects. Discuss whether or not the bitrnap indexes on CustOIuers would help in conlputing the join of Custorners and Prospects on *rating.*
3. Describe the use of a join index to support the join of these two relations with the join condition *custid=prospectid.*

Exercise 25.8 Consider the instances of the Locations, Products, and Sales relations shown in Figure 25.2.

1. Consider the basic join indexes described in Section 25.6.2. Suppose you want to optiInize for the following two kinds of queries: Query 1 finds sa.les in a given city, and Query 2 finds sa.les in a given state. Show the indexes you would create on the example instances shown in Figure 25.2.
2. Consider the bitmapped join indexes described in Section 25.6.2. Suppose you want to optirnize for the following two kinds of queries: Query 1 finds sales in a given city, and Query 2 finds sales in a given state. Show the indexes that you would create on the exanlple instances shown in Figure 25.2.

3. Consider the basic join indexes described in Section 25.6.2. Suppose you want to optimize for these two kinds of queries: Query 1 finds sales in a given city for a given product name, and Query 2 finds sales in a given state for a given product category. Show the indexes that you would create on the exarl1ple instances shown in Figure 25.2.
4. Consider the bitmapped join indexes described in Section 25.6.2. Suppose you want to optirnize for these two kincls of queries: Query 1 finds sales in a given city for a given product narne, and Query 2 finds sales in a given state for a given product category. Show the indexes that you would create on the example instances shown in Figure 25.2.

Exercise 25.9 Consicler the view NurnReservations defined as:

```
CREATE VIEW NumReservations (sid, snarnc, nUlures)
       AS SELECT S.sid, S.snarne, COUNT (*)
          FROM     Sailors S, Reserves R
          WHERE    S.sid = R.sid
          GROUP BY 8.sid, S.sname
```

1. How is the following query, which is intended to find the highest number of reservations nlade by smne one sailor, rewritten using query modification?

```
SELECT   MAX (N.numres)
FROM     NurnReservations N
```

2. Consider the alternatives of cornputing on deluand and view materialization for the preceding query. Discuss the pros and cons of materialization.
3. Discuss the pros and cons of materialization for the following query:

```
SELECT   N.snarlle, MAX (N.numres)
FROM     NumReservations N
GROUP BY N.sname
```

Exercise 25.10 Consider the Locations, Products, and Sales relations in Figure 25.2.

1. To decide whether to rnaterialize a view, what factors do we need to consider?
2. Assurne that we have defined the following lnaterialized view:

```
SELECT   L.state, S.sales
FROM     Locations L, Sales S
WHERE    8.locid=L.locid
```

   (a) Describe what auxiliary infornlatioll the algorithnl for incrernental view rnainte-nance frorn Section 25.10.1 maintains and how this data helps in lnainta.ining the view incrernentally.

   (b) Discuss the pros and cons of ruaterializing this view.

:3. Consider the materialized view in the previous question. Assume that the relations Locations and Sales are stored at Olle site, but the view is rnaterialized on a second site. Why would we ever want to luaintain the view at a second site? Give a concrete exarnple where the view could become inconsistent.

4. ASSUITW that we have defined the following rnaterialized view:

```
SELECT   T.year, I..state, SUM (S.sales)
FROM     Sales 8, 'rirnes '1', Locations L
WHERE    S.tirneid=T.tilneid AND S.locid=L.locid
GROUP BY rr.year, L.state
```

(a) Describe what auxiliary infoflnation the algorithnl for incrernental view rnainte-nance frOIn Section 25.10.1 luaintains, and how this data helps in rnaintaining the view increluentaJly.

(b) Discuss the pros and cons of 11laterializing this view.

BIBLIOGRAPHIC NOTES

A good survey of data warehousing and OLAP is presented in [161], which is the source of Figure 25.10. [686] provides an overview of OLAP and statistical database research, showing the strong parallels between concepts and research in these two areas. The book by Kirnball [436], one of the pioneers in warehousing, and the collection of papers in [(2) offer a good practical introduction to the area. The term OLAP was popularized by Codd's paper [191]. For a recent discussion of the performance of algorithms utilizing bitmap and other nontraditional index structures, see [575].

Stonebraker discusses how queries on views can be converted to queries on the underlying tables through query modification [713]. Hanson cmnpares the perfornlance of query modification versus immediate and deferred view maintenance [365]. Srivastava and Roterll present an analytical model of materialized view maintenance algorithnls [707]. A number of papers discuss how rnaterialized views can be incrementally maintained as the underlying relations are changed. Research into this area has become very active recently, in part because of the interest in *data warehouses,* which can be thought of as collections of views over relations from various sources. An excellent overview of the state of the art can be found in [348], which contains a number of influential papers together with additional rnaterial that provides context and background. The following partial list should provide pointers for further reading: [100, 192, 193, 349, 369, 570, 601, 635, 664, 705, 800].

Gray et al. introduced the CUBE operator [335], and optirnization of CUBE queries and efficient maintenance of the result of a CUBE query have been addressed in several papers, including [12, 94, 216, 367, 380, 451, 634, 638, 687, 799]. Related algorithrns for processing queries with aggregates and grouping are presented in [160, 166]. Rao, Badia, and Van Gucht address the irnplelnentation of queries involving generalized quantifiers such as *a majority of* [618]. Srivastava, Tan, and LUIII describe an access ruethod to support processing of aggregate queries [708]. Shannlugasundaranl et al. discuss how to ruaintain cornpressed cubes for approxirnate answering of aggregate queries in [675].

SQL:1999's support for OLAP, including CUBE and WINDOW constructs, is described in [523]. The windowing extensions are very sirnilar to SQL extension for querying sequence data, called SRQL, proposed in [610]. Sequence queries have received a lot of attention recently. Extending relational systeills, which deal with sets of records, to deal with sequences of records is investigated in [473, 665, 671].

There has been recent interest in one-pass query evaluation algorithnls and database rnanagernent for data streaIns. A recent survey of data rnanagernent for data streams and algorithrns for data stream processing can be fonnd in [49J. Exarnples include quantile and order-statistics cOlnputation [340, 50G], estirnating frequency rnornents and join sizes [34, 35], estirnating correlated aggregates [310], rllultidirnensionaJ regression analysis [173], and cornputing one-dirnensional (i.e., single-attribute) histograrns and Haar wavelet clecmnpositioI1s [319, 345].
Other work includes techniques for incrementally IllElintaining equi-depth histograms [313] and Baal' wavelets [515], rnaintaining sarnples and siluplc statistics over sliding \vindows [201],

as well as general, high-level architectures for stream database systenlS [50]. Zdonik et al. describe the architecture of a database systern for Hl0nit;oring data streaU1S [795]. A language infrastructure for developing data streaIll applications is described by Cortes et al. [199].

Carey and Kossrnann discuss how to evaluate queries for which only the first few answers are desired [135, 1:36]. Donjerkovic and Ralnakrishnan consider how a probabilistic approach to query optiInization call be applied to this probleul [229]. [120] compares several strategies for evaluating Top N queries. Hellerstein et al. discuss how to return approxiInate answers to aggregate queries and to refine thern 'online.' [47, 374]. This work has been extended to online cOlnputation of joins [354], online reordering [617] and to adaptive query processing [48].

There has been recent interest in approximate query answering, where a small synopsis data structure is used to give fast approxiruate query answers with provable perforrnance guarantees [7, 8, 61, 159, 167, 314, 759].

26

DATA MINING

- What is data mining?
- What is lliarket basket analysis? What algorithms are efficient for counting co-occurrences?
- What is the a priori property and why is it important?
- What is a Bayesian network?
- What is a classification rule? What is a regression rule?
- What is a decision tree? How are decision trees constructed?
- What is clustering? What is a salllple clustering algorithln?
- What is a similarity search over sequences? How is it implmuented?
- How can data mining models be constructed increluentally?
- What are the new mining challenges presented by data strealllS?
- Key concepts: data nlining, KDD process; market basket analysis, co-occurrence counting, association rule, generalized association rule; decision tree, classification tree; clustering; sequence similarity search; incrernental model maintenance, data streanls, block evolution

1'he secret of success is to know sornething nobody else knows.

—Aristotle Onassis

Data luining consists of finding interesting trends or patterns in large datasets to guicle decisions about future activities. There is a genera] expectation that data ruining tools should be able to identify these patterns in the data with minirnal user input. The patterns identified by such tools can give a data analyst useful and unexpected insight that can be Illore carefully investigated subsequently, perhaps using other decision support tools. In this chapter, we discuss several widely studied data luining tasks. COllunercial tools are available for each of these tasks frorll major vendors, and the area is rapidly gTowing in ilnportance as these tools gain acceptance in the user cornrnunity.

We start in Section 26.1 by giving a short introduction to data mining. In Section 26.2, we discuss the irnportant task of counting co-occurring items. In Section 26.3, we discuss how this task arises in data mining algorithms that discover rules froln the data. In Section 26.4, we discuss patterns that represent rules in the forln of a tree. In Section 26.5, we introduce a different data rnining task, called *clustering,* and describe how to find clusters in large datasets. In Section 26.6, we describe how to perform siInilarity search over sequences. We discuss the challenges in rnining evolving data and data streams in Section 26.7. We conclude with a short overview of other data mining tasks in Section 26.8.

26.1 INTRODUCTION TO DATA MINING

Data nlining is related to the subarea of statistics called *exploratory data analysis,* which has siruilar goals and relies on statisticalrueasures. It is also closely related to the subareas of artificial intelligence called *knowledge discovery* and *rnachine learning.* The important distinguishing characteristic of data rnining is that the volume of data is very large; although ideas froln these related areas of study are applicable to data nlining problems, *scalability with respect to data size* is an important new criterion. An algorithm is scalable if the running tirne grows (linearly) in proportion to the dataset size, holding the available systenl resources (e.g., arnount of rnain rnemory and CPU processing speed) constant. Old algorithms must be adapted or new algorithnls developed to ensure scalability when discovering patterns from data.

Finding useful trends in datasets is a rather loose definition of data 111ining: In a certain sense, all database queries can be thought of as doing just this. Indeed, we have a continuurn of ana.lysis and exploration tools with SQL queries at one end, OLAP queries in the rniddle, and data ruining techniques at the other end. SQL queries are constructed! using relational algebra (with sorne extensions), OLAP provides higher-level querying idiorlls based on the rnultidirnensional data rn.odel, and data mining provides the rnost abstract analysis operations. We can think of different data rnining tasks as cornplex 'queries' specified at a high level, with a few parameters that are user-defina.ble, and for which specialized algorithrns are implemented.

SQL/MM: Data Mining SQL/MM: The SQL/MM: Data Mining extension of the SQL:1999 standard supports four kinds of data mining nlodels: *frequent itemsets and association rules*, *clusters of records*, *reg'ression trees*, and *classification trees*. Several new data types are introduced. These data types play several roles. SaIne represent a particular class of model (e.g., `DM_RegressionModel`, `DM_ClusteringModel`); some specify the input parameters for a mining algorithm (e.g., `DM_RegTask`, DM_ClusTask); some describe the input data (e.g., `DM_LogicalDataSpec`, `DM_MiningData`); and sornerepresent the result of executing a rnining algorithm (e.g., `DM_RegResult`, DM_ClusResult). Taken together, these classes and their methods provide a standard interface to data mining algorithms that can be invoked frorn any SQL:1999 database systern. The data mining rnodels can be exported in a standard XML format called Predictive Model Markup **Language** (PMML); models represented using PMML can be hnported as well.

In the real world, data rnining is much more than sirnply applying one of these algorithnls. Data is often noisy or inconlplete, and unless this is understood and corrected for, it is likely that rnany interesting patterns will be rnissed and the reliability of detected patterns will be low. Further, the analyst nlust decide what kinds of rnining algoritlulls are called for, apply them to a well-chosen subset of data sarnples and variables (i.e., tuples and attributes), digest the results, apply other decision support and mining tools, and iterate the process.

26.1.1 The Knowledge Discovery Process

The knowledge discovery and data mining (KDD) process can roughly be separated into four steps.

1. Data Selection: The target subset of data and the attributes of interest are identified by exalnining the entire raw dataset.

2. Data Cleaning: Noise and outliers are relnoved, field values are transfonned to cornrnon units and SOUIC new fields are created by cornbining existing fields to facilitate analysis. The data is typically put into a, relational fonnat, and several tables rnight be cornbined in a *denormalization* step.

3. Data Mining: We apply data rnining algorit1ll11S to extract interesting patterns.

4. Evaluation: The patterns are presented to end-users ııı an understandable fonn, for example, through visualization.

The results of any step in the KDD process lllight lead us back to an earlier step to redo the process with the new knowledge gained. In this chapter, however, we limit ourselves to looking at algoritlnns for SaIne specific data rnining tasks. We do not discuss other aspects of the I(DD process.

26.2 COUNTING CO-OCCURRENCES

\Ve begin by considering the probleln of counting co-occurring iterns, which is rnotivated by problelTIs such as lllarket basket analysis. A market basket is a collection of items purchased by a custOlner in a single customer transaction. A cnstorner transaction consists of a single visit to a store, a single order through a mail-order catalog, or an order at a store on the Web. (In this chapter, we often abbreviate *customer transaction* to *transaction* when there is no confusion with the usual nleaning of *transaction* in a DBMS context, which is an execution of a user program.) A COIlIlllon goal for retailers is to identify items that are purchased together. This inforrnation can be used to improve the layout of goods in a store or the layout of catalog pages.

transid	*custid*	*date*	*item*	*qty*
111	201	5/1/99	pen	2
111	201	5/1/99	ink	1
111	201	5/1/99	milk	3
111	201	5/1/99	juice	6
112	105	6/3/99	pen	1
112	105	6/3/99	ink	1
112	105	6/3/99	milk	1
113	106	5/10/99	pen	1
113	106	5/io/99	Inilk	1
114	201	6/1/99	pen	2
114	201	6/1/99	ink	2
114	201	6/1/99	juice	4
114	201	6/1/99	water	1

Figure 26.1 The Purchases Relation

26.2.1 Frequent Itemsets

We use the Purchases relation shown in Figure 26.1 to illustrate frequent itemsets. The records are shown sorted into groups by transaction. All tuples in a group have the saIne *transid*, and together they describe a custorner transaction, which involves purchases of one or Inore iterns. A transaction occurs on a given date, and the nanle of each purchased itenl is recorded, along with the purchased quantity. ()bserve that there is redundancy in Purchases: It can be decolnposed by storing *transid–custid–date* triples in a separate table and dropping *custid* and *date* froln Purchases; this nlay be how the data is actually stored. However, it is convenient to consider the Purchases relation, as shown in Figure 26.1, to corupute frequent iternsets. Creating such 'denormalized' tables for ease of data rnining is cOIIlllonly done in the data cleaning step of the I(DD process.

By examining the set of transaction groups in Purchases, we can rnake observations of the fornl: "In 75% of the transactions a pen and ink are purchased together." rrhis stateulent describes the transactions in the database. Extrapolation to future transactions should be done with caution, as discussed in Section 26.3.6. Let us begin by introducing the terminology of rnarket basket analysis. An **itemset** is a set of itelTIS. The **support** of an itelnset is the fraction of transactions in the database that contain all the iterus in the iterllset. In our exaIupl.e, the itelllset {pen, ink} has 75% support in Purchases. We can therefore conclude that pens and ink are frequently purchased together. If we consider the itelllset {milk, juice}, its support is only 25%; milk and juice are not purchased together frequently.

Usually the nUlnber of sets of itenlS frequently purchased together is relatively sInall, especially as the size of the itenlsets increases. We are interested in all iterllsets whose support is higher than a user-specified minimUIl1 support called *minsup*; we call such itemsets **frequent itemsets**. For exarnple, if the IIlinirl1Unl support is set to 70%, then the frequent iterllsets in our example are {pen}, {ink}, {nlilk}, {pen, ink}, and {pen, 111ilk}. Note that we are also interested in iternsets that contain only a single iteru since they identify frequently purchased iterl1s.

We show an algorithrn for identifying frequent iterllsets in Figure 26.2. This algorithrn relies on a sirnple yet fundarnentaJ property of frequent iterIlsets:

> **The a Priori Property:** Every subset of a frequent iterllset is also a frequent itelnset.

'fhe algorithnl proceeds iteratively, first identifying frequent iterIlsets 'with just one itcrll. In each subsequent iteration, frequent iterl1sets identified in the previous iteration are extended with another itern to generate larger candidate itcrnsets. By considering only iterllsets obtained by enlarging frequent iternsets, we greatly reduce the nurnber of candidate frequent itcrllsets; this optirnization is crucial for efficient execution. The a priori property guarantees that this optilnizatic)ll is correct; that is, we do not Iniss any frequent iterllsets. A single scan of all transactions (the Purchases relation in our example) suffices to

```
foreach itelll,                                                    Level 1
    check if it is a frequent iternset // appears in > minsup transactions
k=l
repeat          // Iterative, level-wise identification of frequent itelllsets
    foreach new frequent iterllset Ik with k iterlls        // Level k + 1
        generate all iterl1sets lk+l with k + 1 itelllS, lk C Ik+l
    Scan all transactions once and check if
    the generated k + 1-iterIlsets are frequent
    k=k+1
until no new frequent itemsets are identified
```

Figure 26.2 An Algorithm for Finding Frequent Itemsets

determine which candidate iterllsets generated in an iteration are frequent. The algorithm terminates when no new frequent itemsets are identified in an iteration.

'We illustrate the algorithrn on the Purchases relation in Figure 26.1, with *minsup* set to 70%. In the first iteration (Levell), we scan the Purchases relation and deterlllline that each of these one-iterll sets is a frequent iternset: *{pen}* (appears in all four transactions), *{ink}* (appears in three out of four transactions), and *{rnilk}* (appears in three out of four transactions).

In the second iteration (Level 2), we extend each frequent itemset with an additional itenl and generate the following candidate iterIlsets: *{pen, ink}, {pen, milk}, {pen, juice}, {ink, rnilk}, {ink, juice},* and *{rnilk, juice}*. By scanning the Purchases relation again, we deterrnine that the following are frequent ite111sets: *{pen, ink}* (appears in three out of four transactions), and *{pen, rnilk}* (appears in three out of four transactions).

In the third iteration (Level 3), we extend these itelllsets with an additional iteHl and generate the following candidate itcrl1sets: *{pen, ink, **milk**}, {pen, ink, juice},* and *{pen, milk, juice}*. (Observe that *{ink, milk, juice}* is not generated.) A third sca.n of the Purchases relation aJlows us to deterrnine that none of these is a frequent iterTlset.

The sirnple algoritlnll presented here for finding frequent iternsets illustrates the principal feature of Inore sophisticated algorithrns, naruely, the iterative generation and testing of candidate itcrnsets. We consider one irnportant refincrnent of this sirnple algorithrn. Cjenerating candidate iternsets by adding an itCHl to a known frequent iternset is an atterIlpt to lirnit the rnunber of candidate itcrIlsets using the a priori property. rrhe a priori property implies that a candidate iternset can be frequent only if all its subsets are frequent. Thus, we can reduce the nUlnber of candidate iternsets further·—*a priori*, or before scanning the Purchases database—-by checking whether all subsets of a newly generated candidate itcIIlset are frequent. Only if all subsets of a candidate iternset are frequent do we cOlnpute its support in the subsequent database scan. COln-pared to the sirnple algoritlun, this refined algoritlull generates fewer candidate itenlsets at each level and thus reduces the arnount of conlputation perfonned during the database scan of Purchases.

Consider the refined algorithrn on the Purchases table in Figure 26.1 with *minsup*= 70%. In the first iteration (Level 1), we deterrnine the frequent item-sets of size one: *{pen}*, *{ink}*, and *{milk}*. In the second iteration (Level 2), only the following candidate itemsets rernain when scanning the Purchases ta-ble: *{pen, ink}*, *{pen, milk}*, and *{ink, rnilk}*. Since *{juice}* is not frequent, the iterllsets *{pen, juice}*, *{ink, juice}*, and *{rnilk, juice}* cannot be frequent as well and we can elirninate those iterIlsets a priori, that is, without considering therll during the subsequent scan of the Purchases relation. In the third iteration (Level 3), no further candidate itemsets are generated. The iternset *{pen, ink, milk}* cannot be frequent since its subset *{ink, milk}* is not frequent. Thus, the irnproved version of the algorithrll does not need a third scan of Purchases.

26.2.2 Iceberg Queries

We introduce iceberg queries through an exaillple. Consider again the Pur-chases relation shown in Figure 26.1. Assurne that we want to find pairs of custorners and iterns such that the custorner has purchased the item rllore than five thnes. We can express this query in SQL as follows:

```
SELECT    P.custid, P.itern, SUM (P.qty)
FROM      Purchases P
GROUP BY  P.custid, P.itern
HAVING    SUM (P.qty) > 5
```

rrhink about how this query would be evaluated by a relational DBMS. Con-ceptually, for each (*custid*, *item*) pair, we need to check whether the surn of the *qty* field is greater than 5. One approach is to rnake a scan over the Purchases relation and rnaintain running surns for each *(c'Ustid, itern)* pair. T'his is a fea-sible execution strategy as long as the nurnber of pairs is sruaU enough to fit into lIlain rncIIlory. If the nurnber of pairs is larger than rnain Inernory, lnorc expensive query evaluation plans,\vhich involve either sorting or hashing, have to be used.

The query has an irnporta"nt property not exploited by the preceding execution strategy: Even though the Purchases relation is potentially very large and the nurnber of (*custid*, *item*) groups CaJl be huge, the Cyutput of the query is likely to be relatively sInall because of the condition in the HAVING clause. ()nly groups where the custorner has purchased the itCHl Inure than five tiInes appear in the output. For exarllple, there are nine groups in the query over the Purchases relation shown in Figure 26.1, although the output contains only three records. The nurnber of groups is very large, but the answer to the query---·-the tip of the iceberg—is usually very sInan. Therefore, we call such a query an iceberg query. In general, given a relational scherna H. with attributes *A.1. A2, ...*, *Ak,* and *B* and an aggrega,tion function aggr, an iceberg query has the follo\ving structure:

```
SELECT    R.Al, R.A2, ..., R,.Ak, aggr(R.B)
FROM      H,elation H,
GROUP BY  R,.AI, ..., R.Ak
HAVING    aggr(R.B) >= constant
```

Traditional query plans for this query that use sorting or hashing first cornpute the value of the aggregation function for all groups and then elirninate groups that do not satisfy the condition in the HAVING clause.

Cornparing the query with the probleur of finding frequent itenlsets discussed in the previous section, there is a striking sirnilarity. Consider again the Purchases relation shown in Figure 26.1 and the iceberg query froIn the beginning of this section. We are interested in (*custid*, *itern)* pairs that have SUM (P.qty) > 5. lJsing a variation of the a priori property, we can argue that we only have to consider values of the *c'Ust'id* field where the custorner has purchased at least five it-eurs. We can generate such iterns through the following query:

```
SELECT    P.cllstid
FROM      Purchases P
GROUP BY  P.cllstid
HAVING    SUM (P.qty) > 5
```

Sirnilarly, we can restrict the candidate values for theitern field through the following query:

```
SELECT    P.itern
FROM      Purchases P
GROUP BY  P.iteul
HAVING    SUM (P.qty) > 5
```

If we restrict the corrlputation of the original iceberg query to (*custid*, *'itern)* groups where the field values are in the output of the previous t\vo queries, we elirninate a large nUlllber of (*custid*, *item*) pairs a priori. So, a possible evaluation strategy is to first COlnpute candidate values for the *custid* and *item* fields, and use eornbinations of only these values in the evaluation of the original iceberg query. We first generate candidate field values for individual fields and use only those values that survive the a priori pruning step as expressed in the two previous queries. Thus, the iceberg query is arnenable to the salIle bottorn-up evaluation strategy used to find frequent iternsets. In particular, we can use the a priori property as follows: We keep a counter for a group only if each individual cOlnponent of the group satisfies the condition expressed in the HAVING clause. The perfonnance irnprovernents of this alternative evaluation strategy over traditional query plans can be very significant in practice.

Even though the botto111-UP query processing strategy elinlinates lnany groups a priori, the nlunber of (*custid*, *itern)* pairs can still be very large in practice; even larger than Inain lllernory. Efficient strategies that use sall1pling and lllore sophisticated hashing techniques have been developed; the bibliographic notes at the end of the chapter provide pointers to the relevant literature.

26.3 MINING FOR RULES

Many algorithrIls have been proposed for discovering various fonns of rules that succinctly describe the data. We now look at some widely discussed fonns of rules and algorithnls for discovering thenl.

26.3.1 Association Rules

We use the Purchases relation shown in Figure 26.1 to illustrate association rules. By examining the set of transactions in Purchases, we can identify rules of the forrn:

$$\{pen\} \Rightarrow \{ink\}$$

This rule should be read as follows: "If a pen is pUfcha.sed in a transaction, it is likely that ink is also be purchased in that transaction." It is a staternent that describes the transactions in the database; extrapolation to future transactions should be done with caution, as discussed in Section 26.3.6. More generally, an association rule has the forIn $LHS \Rightarrow RHS$, where both $LIIS$ and RHS are sets of iterns. The interpretation of such a, rule is that if every itern in $LIIS$ is purchased in a transaction, then it is likely that the iterIlS in RHS are purchased as well.

rThere are two important measures for (1,n association rule:

- Support: The support for a set of iterns is the percentage of transa,ctions that contain all these iterIls. The support for a rule $LIIS \Rightarrow RHS$ is the support for the set of itenlS *LHS* *Rf!S*. For exalnple, consider the rule *{pen}* ⇒ *{ink}*. The support of this rule is the support of the itenlset *{pen, ink}*, which is 75%.

- Confidence: Consider transactions that contain all iterIls in *LHS*. The confidence for a rule *LlIS* ⇒ *RHS* is the percentage of such transactions that also contain all iterIls in *RHS*. More precisely, let *sup(LHS)* be the percentage of transactions that contain *LllS* and let *s'up(LliS* ∪ *RHS)* be the percentage of transactions that contain both *LllS* and *RHS*. rrhen the confidence of the rule *LHS* ⇒ *RHS* is *sup(LHSU RIIS) / sup(LHS)*. The confidence of a rule is an indication of the strength of the rule. As an exalnple, consider again the rule *{pen}* ⇒ *{ink}*. The confidence of this rule is 75%; 75% of the transactions that contain the itenlset *{pen}* also contain the iternset *{ink}*.

26.3.2 An Algorithm for Finding Association Rules

A user can ask for all association rules that have a specified minimum support *(minsup)* and mininlum confidence *(rninconf)*, and various algorithrns have been developed for finding such rules efficiently. These algorithms proceed in two steps. In the first step, all frequent itemsets with the user-specified minimum support are computed. In the second step, rules are generated using the frequent itemsets as input. We discussed an algorithm for finding frequent iternsets in Section 26.2; we concentrate here on the rule generation part.

Once frequent iteulsets are identified, the generation of all possible candidate rules with the user-specified minirnum support is straightforward. Consider a frequent iternset X with support *sx* identified in the first step of the algorithrn. To generate a rule fronl X, we divide *X* into two iternsets, *LHS* and *RJIS*. The confidence of the rule *LllS* ⇒ *RHS* is *Sx / SLIIS*, the ratio of the support of *X* and the support of *LHS*. Frorn the a priori property, we know that the support of *LllS* is larger than *rninsup*, and thus we have C0111puted the support of *L1IS* during the first step of the algoritlnn. We can cornpute the confidence values for the candidate rule by calculating the ratio support(X)/support(LlIS) and then check how the ratio cornpares to *minconf*.

In general, the expensive step of the algorithnl is the cornputation of the frequent itenlsets, and lnany different algorithrns have been developed to perfonn this step efficiently. Rule generation —given that all frequent itcrl1sets have been identified— is straightforward.

In the rest of this section, we discuss sOlne generalizations of the problern.

26.3.3 Association Rules and ISA Hierarchies

In rnany cases, an **ISA** hierarchy or category hierarchy is iInposed on the set of iterl1s. In the presence of a hierarchy, a transaction contains, for each of its iteuls, irnplicitly all the iteln's ancestors in the hierarchy. For example, consider the category hierarchy shown in Figure 26.3. Given this hierarchy, the Purchases relation is conceptually enlarged by the eight records shown in Figure 26.4. rrhat is, the Purchases relation has all tuples shown in Figure 26.1 in addition to the tuples shown in Figure 26.4.

The hierarchy allows us to detect relationships between iterns at different levels of the hierarchy. As an exarnple, the support of the itemset *{ink, juice}* is 50%, but if we replace *juice* with the more general category *beverage*, the support of the resulting itemset *{ink, beverage}* increases to 75%. In general, the support of an itemset can increase only if an item is replaced by one of its ancestors in the ISA hierarchy.

Assulning that we actually physically add the eight records shown in Figure 26.4 to the Purchases relation, we can use any algorithm for computing frequent itemsets on the augmented database. Assuming that the hierarchy fits into rnain memory, we can also perforln the addition on-the-fly while we scan the database, as an optimization.

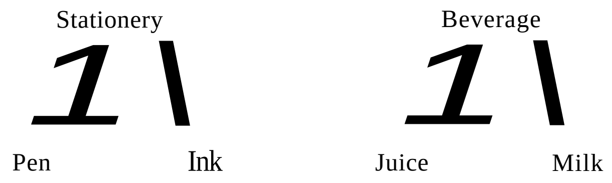

Figure 26.3 An ISA Category Taxonomy

transid	*custid*	*date*	*item*	*qty*
111	201	5/1/99	stationery	3
111	201	5/1/99	beverage	9
112	105	6/3/99	stationery	2
112	105	6/3/99	beverage	1
113	106	5/10/99	stationery	1
113	106	5/10/99	beverage	1
114	201	6/1/99	stationery	4
114	201	6/1/99	beverage	5

Figure 26.4 Conceptual Additions to the Purchases Relation with ISA Hierarchy

26.3.4 Generalized Association Rules

Although association rules have been most \videly studied in the context of market basket analysis, or analysis of cllstorner transactions, the concept is mol'e general. Consider the Purchases relation as shown in Figure 26.5, grouped by *custid*. By exanlining the set of custorner groups, we can identify association rules such as {pen} ⇒ {rnilk}. rThis rule should now be read as follows: "If a pen is purchased by a custorner, it is likely that Inilk is also be purchased by that custcuner." In the Purchases relation shown in Figure 26.5, this rule has both support and confidence of 100%.

transid	*custid*	*date*	*item*	*qty*
112	105	6/3/99	pen	1
112	105	6/3/99	ink	1
112	105	6/3/99	milk	1
113	106	5/10/99	pen	1
113	106	5/10/99	rnilk	1
114	201	5/15/99	pen	2
114	201	5/15/99	ink	2
114	201	5/15/99	juice	4
114	201	6/1/99	water	1
111	201	5/1/99	pen	2
111	201	5/1/99	ink	1
III	201	5/1/99	rnilk	3
111	201	5/1/99	juice	6

Figure 26.5 The Purchases Helation Sorted on Customer ID

Similarly, we can group tuples by date and identify association rules that describe purchase behavior on the same day. As an exalnple consider again the Purchases relation. In this case, the rule {pen} ⇒ {rnilk} is now interpreted as follows: "On a day when a pen is purchased, it is likely that luilk is also be purchased."

If we use the *date* field as grouping attribute, we call consider a rnore general problem called calendric rnarket basket analysis. In calendric rnarket basket analysis, the user specifies a collection of calendars. A, calendar is any group of dates, such as *every Sunday in the year 1999*, or *every first of the month*. A rule holds if it holds on every day in the calendar. Given a calendar, we can cornpute a.ssociatiol1 rules over the set of tuples \vhose *date* field falls within the calendar.

By specifying interesting calendars, we can identify rules that rnight not have enough support and confidence with respect to the entire database but have enough support and confidence on the subset of tuples that fall within the calendar. On the other hand, even though a rule rnight have enough support and confidence \vith respect to the c0l11plete database, it Inight gain its support only £1'0111 tuples that fall within a calendar. In this case, the support of the rule over the tuples within the calendar is significantly higher than its support with respect to the entire database.

As an exarnple, consider the Purchases relation with the calendar *every first of the month*. \Vithin this calendar, the association rule *pen* $\Rightarrow$ *ju:ice* has support and confidence of 100%, whereas over the entire Purcha.ses relation, this rule only has 50% support. On the other hand, within the calendar, the rule *pen* $\Rightarrow$ *milk* has support of confidence of 50%, whereas over the entire Purchases relation it has support and confidence of 75%.

More general specifications of the conditions that rIlust be true within a group for a rule to hold (for that group) have also been proposed. We rnight want to say that all items in the *LHS* have to be purchased in a quantity of less than two itelTIS, and all itenls in the *RHS* rnust be purchased in a quantity of more than three.

lJsing different choices for the grouping attribute and sophisticated conditions as in the preceding exarnples, we can identify rules Inore cornplex than the basic association rules discussed earlier. These Inore cornplex rules, nonetheless, retain the essential structure of an association rule as a condition over a group of tuples, with support and confidence rneasures defined as usual.

26.3.5 **Sequential Patterns**

Consider the Purchases relation sho\vn in Figure 26.1. Each group of tuples, having the sarne *custid* value, can be thought of as a *sequence* of transactions ordered by *date*. rrhis allows us to identify frequently arising buying patterns over tirne.

We begin b,Y introducing the concept of a sequence of itel11sets. Each transaction is represented by a set of tuples, and by looking at the values in the *item* colurnn, we get a set of iterns purchased in that transaction. Therefore, the sequence of transactions associated with a cllstorner corresponds naturally to a sequence of itelnsets purchased by the custorner. For exalnplc, the sequence of purchases for cllstorner 201 is $\langle\{$*pen, ink, milk, juice*$\}, \{$*pen, ink, juice*$\}\rangle$.

A subsequence of a sequence of iternsets is obtained by deleting one or 11101'e itcrnsets, and is also a sequence of itenlsets. We say that a sequence $\langle a_1, \ldots, a_{rn} \rangle$ is contained in another sequence S if S has a subsequence $(b_t, \ldots, bln)$ such that $a_i \subseteq b_i$, for $1 \leq i \leq rn$. Thus, the sequence ⟨{*pen*}, {*ink, rnilk*}, {*pen, ju'ice*}) is contained in ⟨{*pen, link*}, {*shir·t*}, {*ju'ice, ink, milk*}, {*juice, pen, milk*}⟩. Note that the order of itenlS within each iterllset does not rnatter. However, the order of iterllsets does lllatter: the sequence ({*pen*}, {*ink, rn'ilk*}, {*pen, juice*}⟩ is not contained in ⟨{*pen, ink*}, {*shirt*}, {*juice, pen, rnilk*}, {*juice, milk, 'ink*}).

The support for a sequence *S* of iternsets is the percentage of custorner sequences of which 8 is a subsequence. The problenl of identifying sequential patterns is to find all sequences that have a user-specified rllinimurll support. A sequence *(aI, a2, a3, ... ,am)* with minimurn support tells us that custorners often purchase the itelns in set a1 in a transaction, then in sonle subsequent transaction buy the itcrlls in set *a2,* then the items in set *a3* in a later transaction, and so on.

Like association rules, sequential patterns are staternents about groups of tuples in the current database. Cornputationally, algorithms for finding frequently occurring sequential patterns resernble algorithrns for finding frequent itemsets. Longer and longer sequences with the required rninirnum support are identified iteratively in a nlanner very similar to the iterative identification of frequent iternsets.

26.3.6 The Use of Association Rules for Prediction

Association rules are widely used for prediction, but it is inlportant to recognize that such predictive use is not justified without additional analysis or dornain knowledge. Association rules describe existing data accurately but can be misleading when used naively for prediction. For exaruple, consider the rule

$$\{pen\} \Rightarrow \{ink\}$$

The confidence associated with this rule is the conditional probability of an ink purchase given a pen purchase *over the given database;* that is, it is a *descriptive* rueasure. We rnight use this rule to guide future sales prornotions. For exalllple, we rnight offer a discount on pens to increase the sales of pens and, therefore, aIso increase sales of ink.

Flowever, such a prorllotion assumes that pen purchases are good indicators of ink purchases in *future* custC)Iuer transactions (in addition to transactions in the current database). This assumption is justified if there is a *causal link* between pen purchases and ink purchases; that is, if buying pens causes the buyer to also buy ink. Ifowever,we can infer association rules\vith high support and confidence in sOlnc situations where there is no causal link between *L118* and *RIIS*. For exarnple, suppose that pens are ahvays purchased together with pencils, perhaps because of customers' tendency to order writing instrulllents together. We would then infer the rule

$$\{pencil\} \Rightarrow \{ink\}$$

with the saBle support and confidence as the rule

$$\{pen\} \Rightarrow \{ink\}$$

However, there is no causal link between pencils and ink. If we prornote pencils, a custolner who purchases several pencils due to the pronlotion has no reason to buy Inore ink. Therefore, a sales prolnotion that discounted pencils in order to increase the sales of ink would fail.

In practice, one would expect that, by exallllnlng a large database of past transactions (collected over a long tirne and a variety of circumstances) and restricting attention to rules that occur often (i.e., that have high support), we rninirnize inferring lnisleading rules. However, we should bear in rnind that nlisleading, noncausal rules lnight still be generated. Therefore, we should treat the generated rules as possibly, rather than conclusively, identifying causal relationships. Although association rules do not indicate causal relationships between the *LHS* and *RHS*, we elllphasize that they provide a useful starting point for identifying such relationships, using either further analysis or a dornain expert's judgrnent; this is the reason for their popularity.

26.3.7 Bayesian **Networks**

Finding causal relationships is a challenging task, as we saw in Section 2G.3.6. In general, if certain events are highly correlated, there are rnany possible explanations. For exalnple, suppose that pens, pencils, and ink are purchased together frequently. It rnight be that the purchase of one of these itelIlS (e.g., ink) depends causally on the purchase of another itern (e.g., pen). ()r it Blight be that the purchase of one of these iterns (e.g., pen) is strongly correlated with the purchase of another (e.g., pencil) because of sorne underlying phenornenon (e.g., users' tendency to think about \vriting instrulnents together) that causally influences both purchases. How can we identify the true causal relationships that hold between these events in the real world?

One approach is to consider each possible cOlnbination of causal relationships arnong the varial)les or events of interest to us and evaluate the likelihood of each cornbination on the basis of the data available to us. If we think of each cornbination of causal relationships as a *model* of the real world underlying the collected data, we can assign a score to each ruode! by considering how consistent it is (in terms of probabilities, 'with senne sin1plifying assumptions) with the observed data. Bayesian networks are graphs that can be used to describe a class of such Il1odels, with one node per variable or event, and arcs between nodes to indicate causality. For exarnpIe, a good Iuodel for our running exarnpIe of pens, pencils, and ink is shown in Figure 26.6. In general, the nurnber of possible Inodels is exponential in the nurnber of variables, and considering all rnodels is expensive, so SOUle subset of all possible rnodels is evaluated.

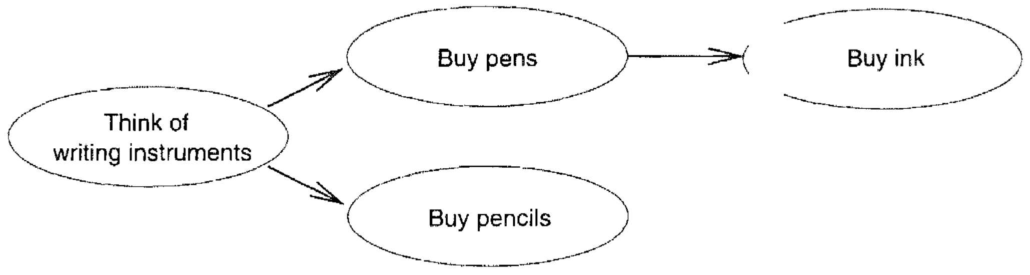

Figure 26.6 Bayesian Network Showing Causality

26.3.8 Classification and Regression Rules

Consider the following view that contains infornation froln a rnailing carnpaign performed by an insurance cornpany:

InsuranceInfo(*age:* integer, *cartype:* string, *highrisk:* boolean)

The Insurancelnfo vie\v has infornation about current cllston1ers. Each record contains a cllstolner's age and type of car as well as a flag indicating whether the person is considered a high-risk custorner. If the flag is true, the cllstorner is considered high-risk. We would like to use this information to identify rules that predict the insurance risk of new insurance applicants whose age and car type are known. :For exarnple, one such rule could be: "If *age* is between 16 and 25 and *cartype* is either Sports or.Truck, then the risk is high."

Note that the rules we want to find have a specific structure.vVe are not interested in rules that predict the age or type of car of a person: we are interested only in rules that predict the insurance risk. Thus, there is one designated attribute whose value we wish to predict, and we call this attribute the dependent attribute. rrhe other attributes aTe called predictor attributes. In our example, the dependent attribute in the Insurancelnfo vic\v is the *highrisk* attribute arld the predictor attributes are *age* and *cartype*. The general foru1 of the types of rules we want to discover is

$$P_1(X_1) \wedge P_2(X_2) \ldots \wedge P_k(X_k) \Rightarrow Y = c$$

The predictor attributes $X_1, \ldots, X_k$ are used to predict the value of the dependent attribute Y. Both sides of a rule can be interpreted as conditions on fields of a tuple. The $Pi(X_i)$ are predicates that involve attribute X_i. The fornl of the predicate depends on the type of the predictor attribute. We distinguish two types of attributes: numerical and categoricaL For numerical attributes, we can perforIn nurnerieal cornputations, such as cornputing the average of two values; whereas for categorical attributes, the only allowed operation is testing whether two values are equal. In the InsuranceInfo view, *age* is a nUlllerical attribute whereas *cartype* and *highrisk* are categorical attributes. Returning to the forrn of the predicates, if X_i is a nUlllerical attribute, its predicate P_i is of the forln $li \leq X_i \leq hi$; if X_i is a categorical attribute, Pi is of the forlll X_i E {Vl, ... ,Vj}.

If the dependent attribute is categorical, we call such rules classification rules. If the dependent attribute is nurnerical, we call such rules regression rules.

For exarnple, consider again our exaInple rule: "If *age* is between 16 and 25 and *caTtype* is either Sports or Truck, then *highrisk* is true." Since *highrisk* is a categorical attribute, this rule is a classification rule. We can express this rule fonnally as follows:

$$(16 \leq age \leq 25) \wedge (cartype \text{ E } \{\text{Sports, Truck}\}) \Rightarrow highri8k = \text{true}$$

We can define support and confidence for classification and regression rules, as for association rules:

- Support: ffhe support for a condition C is the percentage of tuples that satisfy C. The support for a rule $C1 \Rightarrow C{:}2$ is the support for the condition CI/\ *C2*.

- Confidence: Consider those tuples that satisfy condition $C1$. The confidence for a rule Cl $\Rightarrow C2$ is the percentage of such tuples that also satisfy condition $C2$.

As a further generalization, consider 1,118 right-hand side of a classification or regression rule: Y =. c..Each rule predicts a value of Y for a given tuple based on the values of predictor attributes Xl, ... ,*Xk*. We can consider rules of the fonn

$$P_1(X_1) \wedge \ldots \wedge P_k(X_k) \Rightarrow Y = f(X_1, \ldots, X_k)$$

where f is sonlC function. We do not discuss such rules further.

Classification and regression rules differ fr0111 clssociation rules by considering continuous and categorical fields, rather than only one field that is set-valued. Identifying such rules efficiently presents a new set of challenges; we do not discuss the general case of discovering such rules. We discuss a special type of such rules in Section 26.4.

Classification and regression rules have many applications. Exarnples include classification of results of scientific experirnents, where the type of object to be recognized depends on the measurements taken; direct lllail prospecting, where the response of a given customer to a prolnotion is a function of his 01' her inCOlue level and age; and car insurance risk assesslnent, where a customer could be classified as risky depending on age, profession, and car type. Example applications of regression rules include financial forecasting, where the price of coffee futures could be SOlne function of the rainfall in Colornbia a month ago, and Inedical prognosis, where the likelihood of a tUlnor being cancerous is a function of Illeasured attributes of the tUlnor.

26.4 TREE·STRUCTURED RULES

In this section, we discuss the problem of discovering classification and regression rules from a relation, but we consider only rules that have a very special structure. The type of rules we discuss can be represented by a tree, and typically the tree itself is the output of the data mining activity. Trees that represent classification rules are called classification trees or decision trees and trees that represent regression rules are called regression trees

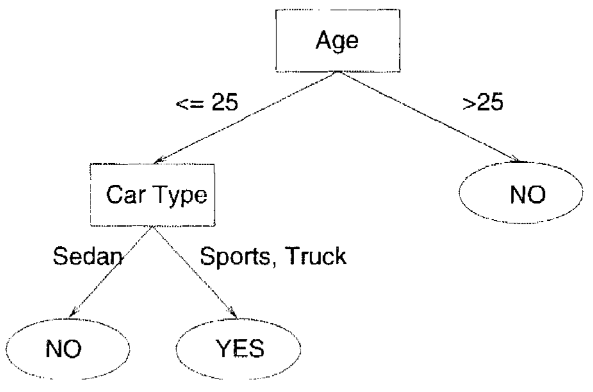

Figure 26.7 Insurance Risk Example Decision Tree

As an exalnple, consider the decision tree ShO\Vll in Figure 26.7. Each path froln the root node to a leaf node represents one classification rule. For example, the path fronl the root to t1le leftrnost leaf node represents the classification rule: "If a person is 25 years or .younger and drives a sedan, then he or she is likely to have a low insurance risk." The path fforn the root to the right-Inost leaf node represents the classification rule: "If a person is older than 25 years, then he or she is likely to have a low insurance risk."

Tree-structured rules are very popular since they are easy to interpret. Ease of understanding is very hllportant because the result of any data rninillg activity needs to be cOlllprehensible by nonspecialists. In addition, studies have shown that, despite Ihnitations in structure, tree-structured rules are very accurate. There exist efficient algorithrl1s to construct tree-structured rules fronl large databases. We discuss a sample algorithrIl for decision tree construction in the rernainder of this section.

26.4.1 Decision Trees

A decision tree is a graphical representation of a collection of classification rules. Given a data record, the tree directs the record frOIn the root to a leaf. Each internal node of the tree is labeled with a predictor attribute. This attribute is often called a **splitting attribute**, because the data is 'split' based on conditions over this attribute. The outgoing edges of an internal node are labeled with predicates that involve the splitting attribute of the node; every data record entering the node must satisfy the predicate labeling exactly one outgoing edge. T'he cornbined information about the splitting attribute and the predicates on the outgoing edges is called the **splitting criterion** of the node. A node with no outgoing edges is called a **leaf node**. Each leaf node of the tree is labeled with a value of the dependent attribute. We consider only binary trees where internal nodes have two outgoing edges, although trees of higher degree are possible.

Consider the decision tree shown in Figure 26.7. The splitting attribute of the root node is *age*, the splitting attribute of the left child of the root node is *cartype*. The predicate on the left outgoing edge of the root node is $age \leq 25$, the predicate on the right outgoing edge is $age > 25$.

"'e can no\v aBsociate a classification rule with each leaf node in the tree as follows. Consider the path frorH the root of the tree to the leaf node. Each edge on that path is labeled with a predicate. 'The conjunction of all these predicates rnakes up the left-hand side of the rule. rrhe value of the dependent attribute at the leaf node rnakesup the right-ha,nd side of the rule. Thus, the deeision tree represents a collection of classification rules, OIle for each leaf node.

A decision tree is usually constructed in t\VO phases. In phase one, the **growth phase**, an overly large tree is constructed. This tree represents the records in the input database very accurately; for exaluple, the tree rnight contain leaf nodes for inclividual records frorn the input database. Tn phase t\VO, the **pruning phase**, the final size of the tree is deterrnined. The rules represented by the tree constructed in phase one are usuall:y overspecialized. By reducing the size of the tree, we generate a srnaller nUlnber of lllore general rules that are better than a very large nUlllbcr of very specialized rules. Algorithrns for tree pruning are beyond our scope of discussion here.

Classification tree algorithrlls build the tree greedily top-down in the following way. At the root node, the database is exarnined and the locally 'best' splitting criterion is cornputed. rrhe database is then partitioned, according to the root node's splitting criterion, into two parts, one partition for the left child and one pa,rtition for the right child. The algoritlull then recurses on each child. rrhis schcrua is depicted in Figure 26.8.

```
Input: !loden, partition D, split selection ruethod S
Output: decision tree for D rooted at node n

Top-Down Decision Tree Induction Schema:
BuildTree(Node 11, data partition D, split selection rnethod S)
(1)  Apply S to D to find the splitting criterion
(2)  if (a good splitting criterioll is found)
(3)     Create two children nodes n1 and n2 of n
(4)     Partition D into D1 and D2
(5)     BuildT'ree(nl, D1, S)
(6)     Build'Tree(n2, D2, S)
(7)  endif
```

Figure 26.8 Decision Tree Induction Schema

The splitting criterion at a node is found through application of a split selection method. A split selection rnethod is an algorithrIl that takes as input (part of) a relation and outputs the locally 'best' splitting criterion. In our exarnple, the split selection rnethod exarnines the attributes *cartype* and *age*, selects one of thern as splitting attribute, and then selects the splitting predicates. IVlany different, very sophisticated split selection rnethods have been developed; the references provide pointers to the relevant literature.

26.4.2 An Algorithm to Build Decision Trees

If the input database fits into rna,in Inernory, we can directly follow th.e classification tree induction schcrna shown in Figure 26.8. How can we construct decision trees when the input relation is larger than rnain rncrJlory? In this case, step (1) in Figllre 26.8 fails, since the input database does not fit in Inenl0ry. But we can rnake one irnportant observation about split selection Inethods that helps us to reduce the rnain rnerllory requircluents.

Consider a node of the decision tree. The split selection rnethod has to Inake two decisions after exarllining the partition at that node: It has to select the splitting attribute, and it has to select the splitting predicates for tIle outgo-

age	*cartype*	*highrisk*
23	Sedan	false
30	Sj)orts	false
36	Sedan	false
25	Truck	true
30	Sedan	false
23	Truck	true
30	Truck	false
25	Sports	true
18	Sedan	false

Figure 26.9 The Insurancelnfo Relation

ing edges. After selecting the splitting criterion at a node, the algorithrn is recursively applied to each of the children of the node. Does a split selection rnethod actually need the cornplete database partition as input? Fortunately, the answer is no.

Split selection rnethods that cornpute splitting criteria that involve a single predictor attribute at each node evaluate each predictor attribute individually. Since each attribute is exarnined separately, we can provide the split selection rnethod with aggregated inforulation about the database instead of loading the cornplete database into rnain rnenlory. Chosen correctly, this aggregated inforrnation enables us to cornpute the same splitting criterion as we would obtain by exarnining the conlplete database.

Since the split selection rnethod exanlines all predictor attributes, we need aggregated inforrnation about each predictor attribute. We call this aggregated inforrnation the AVe set of the predictor attribute. The AVe set of a predictor attribute X at noden is the projection of n's database partition onto X and the dependent attribute where counts of the individual values in the dorllain of the dependent attribute are aggregated. (AVC stands for Attribute-Value, Class label, because the values of the dependent attribute are ofterl called class labels.) For example, consider the Insurancelnfo relation as shown in Figure 26.9. rrhe AVe set of the root node of the tree for predictor attribute *age* is the result of the following database query:

```
SELECT   R.age, Il.highrisk, COUNT (*)
FROM     Insurancelnfo R
GROUP BY R.age, R.highrisk
```

The AVe set for the left child of the root node for predictor attribute *cartype* is the result of the following query:

```
SELECT    R.cartype, R.highrisk, COUNT (*)
FROM      Insurancelnfo R
WHERE     R.age <= 25
GROUP BY  R.cartype, R.highrisk
```

The t\VO AVC sets of the root node of the tree are shown in Figure 26.10.

Car type	highrisk	
	true	false
Sedan	0	4
Sports	1	1
Truck	2	1

Age	highrisk	
	true	false
18	0	1
23	1	1
25	2	0
30	0	3
36	0	1

Figure 26.10 AVe Group of the Root Node for the Insurancelnfo Relation

We define the AVe group of a node *n* to be the set of the AVe sets of all predictor attributes at node *n*. Our exarnple of the Insurancelnfo relation has two predictor attributes; therefore, the AVe group of any node consists of two AVe sets.

How large are AVe sets? Nate that the size of the AVe set of a predictor attribute *X* at node *n* depends only on the nurnber of distinct attribute values of *X* and the size of the dornain of the dependent attribute. For exarnple, consider the AVe sets shown in Figure 26.10. The AVe set for the predictor attribute *cartype* has three entries, and the AVe set for predictor attribute *age* has five entries, although the Insuraneelnfo relation as shown in Figure 26.9 has nine records. For large databases, the size of the AVe sets is independent of the nurnber of tuples in the database, except if there are attributes with very large dOITlains, for exarnple, a real-valued field recorded at a very high precision with rnany digits after the decirnal point.

If we 1nake the sirnplifying assurnption that all the AVe sets of the root node together fit into rnain rnernory, then we can construct decision trees frOTH very large databases as follo\vs: We rnake a scan over the database and construct the AVe group of the root node in rneIllory. Then we run the split selection rnethod of our choicc\vith t11e AVC group as input. After the split selection 1netllod cornputes the splitting attribute and the splitting predicates on the outgoing nodes, we partition the database and recurS8. Note that this algorithrIl is very similar to the original algorithrn shown in Figure 26.8; the only rnodification necessary is shown in Figure 26.11. In additioll, this algoritlll11 is still independent of the actual split selection rnethod involved.

```
Input: node $n$, partition D, split selection 111ethod $S$
Output: decision tree for $D$ rooted at node $n$

Top-Down Decision Tree Induction Schenla:
BuHdTree(Node $n$, data partition $D$, split selection method $S$)
(la) Make a scan over D and construct the AVe group of $n$ in-nlCIIlory
(1b) Apply $S$ to the AVe group to find the splitting criterion
```

Figure 26.11 Classification rn-ee Induction Refinement with AVe Groups

26.5 CLUSTERING

In this section we discuss the clustering problem. The goal is to partition a set of records into groups such that records within a group are shnilar to each other and records that belong to two different groups are dissimilar. Each such group is called a cluster and each record belongs to exactly one cluster.[1] Sirnilarity between records is Ineasured cOlnputationally by a distance function. A distance function takes two input records and returns a value that is a measure of their silnilarity. Different applications have different notions of similarity, and no one rneasure works for all domains.

As an exarnple, consider the scherna of the Custol11erlnfo view:

CustornerInfo(*age:* int, *salary:* real)

We can plot the records in the view on a two-dil11ensional plane as shown in Figure 26.12. The two coordinates of a record are the values of the record's *salary* and *age* fields. \Ve can visually identify three clusters: Young cllstorners who have low salaries, young cllstorners with high salaries, and older cnstorners with high salaries.

Usnally, the output of a clustering algorithrll consists of a summarized representation of each cluster. The type of sUIrllnarized representation depends strongly on the type and shape of clusters the algoritlull cornputes. For exarnple, assume that we have spherical clusters as in the exalllple shown in Figure 26.12. We can surnrnarize each cluster by its *center* (often also called the *mean*) and its *radius*, which are defined as follo\vs. Given a collection of records $r_1, \ldots, r_n$, their center C and radius R are defined as follows:

$$C = \frac{1}{n}\sum_{i=1}^{n} r_i, \text{ aIIc}^1 R = \sqrt{\frac{I\sum_{i=1}^{n}(r_i - C)}{n}}$$

[1]There are clustering algorithrns that allow overlapping clusters, where a record could belong to several clusters.

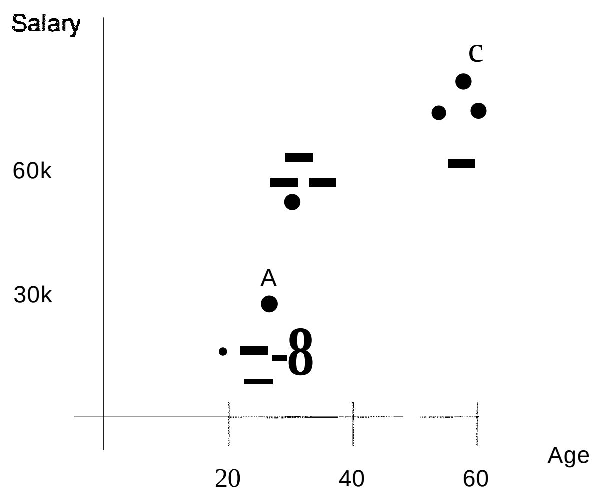

Figure 26.12 Records in CustomerInfo

There are two types of clustering algorithlns. A **partitional** clustering algorithnl partitions the data into k groups such that SOUle criterion that evaluates the clustering quality is optirnized. The nurnber of clusters k is a parameter whose value is specified by the user. A **hierarchical** clustering algorithnl generates a sequence of partitions of the records. Starting with a partition in which each cluster consists of one single record, the algorithrn rnerges two partitions in each step until only one single partition rernains in the end.

26.5.1 A Clustering Algorithm

Clustering is a very old problern, and nurnerous algorithnls have been developed to cluster a collection of records. Traditionally, the nurnber of reeords in the input database was assurned to be relatively slnall and the cornplete database was assurned to fit into Inain rnernory. In this section,we describe a clustering algoritlnn called BIRCH that handles very large databases. The design of BIR,CII reflects the follovving two assumptions:

- The rnunber of records is potentially very large, and therefore we want to rnake only one scan over the database.
- Only a lirnited arnount of rnain rnenlory is available.

A user can set t\VO pararneters to control the BIRCH algoritllln. The first is a thresl10lcl on the arnount of rnain luernory available. This main rncrnory threshold translates into a lllaxirnurn nurnber of cluster SUIJlrnaries k that can be maintained in rncrllory. 'The second pararneter ϵ is an initial threshold for the radius of an,Y cluster. The value of ϵ is an upper bound on the radius of any cluster and controls the nUlnber of clusters that the algorithrn discovers. If ϵ is slnalI, we discover many slnaII clusters; if ϵ is large, we discover very fe\v clusters, each of which is relatively large. We say that a cluster is compact if its radius is s1na11e1' than ϵ.

BIRCH always lTlaintains k or fewer cluster sU1nrnaries (C_i, R_i) in rnain Hlcnl0ry, where C_i is the center of cluster i and R_i is the radius of cluster i. The a1gorithrn ahvays rnaintains cornpact clusters; that is, the radius of each cluster is less than E. If this invariant cannot be rnaintained with the given arIlount of rnain lnerno1'y, ϵ is increased as described next.

The algoritlnl1 reads records frorn the database sequentially and processes the1l1 as follows:

1. Cornpute the distance betVileen record r and each of the existing cluster centers. Let i be the cluster index such that the distance between rand C_i is the srnallest.
2. Cornpute the value of the new radius R_i' of the ith cluster under the assumption that r is inserted into it. If $R_i' \leq \epsilon$, then the ith cluster rernains cornpact, and we assign r to the ith cluster by updating its center and setting its radius to R_i'. If $R_i' > \epsilon$, then the ith cluster would no longer be cOlnpact if we insert r into it. Therefore, we start a new cluster containing only the record r.

The second step presents a problern if we already have the rnaxinnun nurnber of cluster sUIInnaries, k. If we now read a record that requires us to create a new cluster, we lack the rnain rne1nory required to hold its surnrnary. In this case, we increase the radius threshold E-----using SOHle heuristic to detennine the increase---in order to *merge* existing clusters: An increase of ϵ has two consequences. First, existing clusters can accorl1rnodate rnore records, since their rnaxirnurn radius has increased. Second, it Blight be possible to rnerge existing clusters such that the resulting cluster is still cornpact. rrhus, an increase in ϵ usually reduces the l1ulIlber of existing clusters.

The cornplete BIRCH algorithrl1 uses a balanced in-rnernory tree, which is sirnilar to a B+ tree in structure, to quickly identify the closest cluster center for a new record. A description of this data structure is beyond the scope of our discussion.

26.6 SIMILARITY SEARCH OVER SEQUENCES

A lot of inforrnation stored in datal)ases consists of sequences. In this section, we introduce the problern of siInilarity search over a collection of sequences. Our query Inode} is very sirnple: We assurne that the user specifies a query sequence and wants to retrieve all data sequences that are similar to the

Commercial Data Mining Systems: There area number of data ruining products on the rnarket today, such as SASEnterprise Miner, SPSS Clenlcntine, CART froIn Salford Systems, Megaputer PolyAnalyst, ANGOSS I<nowledgeStudio. We highlight two that have strong database ties.

IBM's Intelligent Miner offers a wide range of algorithlns, including association rules, regression, classification, and clustering. The emphasis of Intelligent Miner is on scalability—the product contains versions of all algorithllls for parallel cOlnputers and is tightly integrated with IBM's DB2 database systenl. DB2's object-relational capabilities can be used to define the data Inining classes of SQL/MM. Of course, other data 111ining vendors can use these capabilities to add their own data mining models and algorithms to DB2.

Microsoft's SQL Server 2000 has a component called the Analysis Server that lnakes it possible to create, apply, and lnanage data mining models within the DBMS. (SQL Server's OLAP capabilities are also packaged in the Analysis Server component.) The basic approach taken is to represent a mining rrlodel as a table; clustering and decision tree models are currently supported. The table conceptually has one row for each possible combination of input (predictor) attribute values. The model is created using a staternent analogous to SQL's CREATE TABLE that describes the input on which the model is to be trained and the algorithrn to use in constructing the model. An interesting feature is that the input table can be defined, using a specialized view rnechanisnl, to be a *nested table.* For exalnple,we can define an input table with one row per custolner, where one of the fields is a nested table that describes the eustolner's purchases. The SQL/MM extensions for data ruining do not provide this capability because SQL:1999 does not currently support nested tables (Section 23.2.1). Several properties of attributes, such as whether they are discrete or continuous, can also be specified.

A rnodel is trained by inserting rows into it, using the INSERT cornlnand. It is applied to a new dataset to lnake predictions using a new kind of join called PREDICTION JOIN; in principle, each input tuple is matched with the corresponding tuple in the rnining lllodel to detennine the value of the predicted attribute. Thus, end users can create, train,and apply decision trees and clustering using extended SQL. 'There are also cornrnands to browse rnodels. Unfortnnately, users cannot add new rnodels or new algorithrlns for models, a capability that is supported in the SQL/MlVI proposa.

query sequence. Sinlilarity search is different frorH 'normal' queries in that we are interested not only in sequences that rnatch the query sequence exactly but also those that differ only slightly frorn the query sequence.

We begin by describing sequences and sirnilarity between sequences. A data sequence X is a series of nurnbers X = $\langle x_1, \ldots$ *Xk)*. S0111etirIles X is also called a time series. We call *k* the length of the sequence. A subsequence *Z* = (Zl' ... , $z_j\rangle$ is obtained frolll another sequence X = *(Xl, ... ,Xk)* by deleting nurnbers froln the front and back of the sequence X. ForInally, *Z* is a subsequence of X if *Z1* = *Xi, Z2* = x_{i+1}, *,Zj* = *Zi-tj-l* for SaIne i E {I, ... , *k-j* +I}. Given two sequences X = (Xl, *,Xk)* and Y = *(YIl" .,Yk),* we can define the Euclidean **Darrri** as the distance between the two sequences *as* follows:

$$\|X - Y\| = \sum_{i=1}^{k}(x_i - y_i)^2$$

Given a user-specified query sequence and a threshold pararneter ϵ, our goal is to retrieve all data sequences that are within E-distance of the query sequence.

Sirnilarity queries over sequences can be classified into two types.

- Complete Sequence Matching: The query sequence and the sequences in the database have the sarne length. Given a user-specified threshold paranleter ϵ, our goal is to retrieve all sequences in the database that are within E-distance to the query sequence.
- Subsequence Matching: rrhe query sequence is shorter than the sequences in the database. In this case, we want to find all subsequences of sequences in the database such that the subsequence is within distance ϵ of the query sequence. We do not discuss subsequence rnatching.

26.6.1 An Algorithm to Find Similar Sequences

Given a collection of data sequences, a query sequence, and a distance threshold ϵ, how can we efficiently find all sequences within f-distance of the query sequence?

()ne possibility is to scan the database, retrieve each data sequence, and C0111pute its distance to the query sequence. \Vhile this algorithrn has the rnerit of being sirnple, it always retrieves every data sequence.

Because we consider the conlplete sequence lnatehing problenl, all data sequences and the query sequence have the same length. We can think of this sirnilarity search as a high-dirnensional indexing probleul. Each data sequence and the query sequence can be represented as a point in a k-dirnensionaJ space. Therefore, if we insert all data sequences into a Illuitidirnensional index, we can retrieve data sequences that exactly ll1atch the query sequence by qllerying the index. But since 'we want to retrieve not only data sequences that Inatch the query exactly but also all sequences within (-distance of the query sequence, we do not use a point query as defined by the query sequence. Instead, we query the index 'with a hyper-rectangle that has side-length 2E and the query sequence as center, and we retrieve all sequences that fall within this hyper-rectangle. We then discard sequences that are actually further than ϵ away froln the query sequence.

ITsing the index allows us to greatly reduce the nurnber of sequences we consider and decreases the time to evaluate the sirnilarity query significantly. The bibliographic notes at the end of the chapter provide pointers to further improvernents.

26.7 INCREMENTAL MINING· AND DATA STREAMS

Real-life data is not static, but is constantly evolving through additions or deletions of records. In sorne applications, such as network Inonitoring, data arrives in such high-speed strearns that it is infeasible to store the data for offline analysis. We describe both evolving and strearning data in terlns of a framework called block evolution. In block evolution, the input dataset to the data mining process is not static but periodically updated with a new block of tuples, for exarnple, every day at rnidnight or in a continuous strealn. A block is a set of tuples added siInultaneously to the database. For large blocks, this Inodel captures comrnon practice in rnany of today's data warehouse installations, where updates from operational databases are batched together and perforrned in a block update. For srnall blocks of data—at the extrerne, each block consists of a single record—this rnodel captures strealning data.

In the block evolution rnodel, the database consists of a (conceptually infinite) sequence of data blocks $D_1, [J_2, \ldots$ that arrive at tilnes I, 2, ...,\Vhe1'8 each block D_i consists of a set of records.[2] We call i the *block identifier* of block 13_i. Therefore, at any titHe t, the database consists of a finite sequence of blocks of data $\langle D_1, \cdots, D_t \rangle$ that arrived at tirnes $\{I, 2, \ldots, t\}$. The database at tilne t, \vhic.h we denote by 1)[1, t], is the union of the database at time t - 1 and the block that arrives at tirue t, D_t.

For evolving data, two classes of problerns are of particular interest: rnodel Inaintenance and change detection. The goal of 1110del maintenance is to

2In general, a block specifies records to change or delete, in addition to records to insert. We only consider inserts.

maintain a data rnining ulodcl under insertion and deletions of blocks of data. To incrernentally cornpute the data mining rnodel at time t, which we denote by $M(D[1, t))$, we HUlst consider only $M(D[1, t-1])$ and $.D_t$; we cannot consider the data that arrived prior to time t. Further, a data analyst rllight specify tirne-dependent subsets of $D[1, t]$, such as a window of interest (e.g., all the data seen thus far or last week's data). More general selections are also possible, for exarnple, all weekend data over the past year. Given such selections, we Hlllst incrernentally CCHupute the rnodel on the appropriate subset of $.D[l, t]$ by considering only $[J_t$ and the model on the appropriate subset of $1)[1, t - 1]$. 'Alrnost' incrernental algoritlulls that occasionally exarnine older data rnight be acceptable in warehouse applications, where incrementality is lTIotivated by efficiency considerations and older data is available to us if necessary. This option is not available for high-speed data strearns, where older data may not be available at all.

The goal of change detection is to quantify the difference, in terrns of their data characteristics, between two sets of data and determine whether the change is rneaningful (i.e., statistically significant). In particular, we rnust quantify the difference between the rllodels of the data as it existed at sonle time tl and the evolved version at a subsequent ·tirne $t2$; that is, we Blust quantify the difference between $1\1(D[I, tl])$ and $1\1(D[1, t2])$. We can also measure changes with respect to selected subsets of data. Several natural variants of the problem exist; for exarnple, the difference between $M(D[l, t - 1])$ and $M(D_t)$ indicates whether the latest block differs substantially frorn previously existing data. In the rest of this chapter, we focus on rnodel rnaintenance and do not discuss change detection.

Incrernental rnodel rnaintenance has received rnuch attention. Since the quality of the data rllining rnodel is of utrnost irnportance, incrernental rnodel rnaintena,nce a1gorithrns have concentrated on cornputing exactly the sarne Inode1 as cOlnputed by running the basic rnodel construction algoritlul1 on the union of old and new data. ()ne \videly used scalability technique is localization of changes due to new blocks. For exarnple, for density-based clustering algoritluns, the insertion of a new record affects only clusters in the neighborhood of the record, and thus efficient algorithrlls can *localize* the change to a few clusters and avoid reccHnputing all clusters. As another exarllple, in decision tree construction, we rnight be able to show that the split criterion at a, node of the tree changes only within acceptably srnall confidence intervals when records are inserted, if we assume tha,t the underlying distribution of training records is static.

One-pass rnodel construction over data strearllS has received particular attention, since data arrives and rnust be processed continuously in several ernerging application dCHnains. For exarnple, network installations of large TelecOll1 and Internet service providers have detailed usage inforruation (e.g., eall-detail-records, router packet-flow and trace data) froln different parts of the underlying network that needs to be continuously analyzed to detect interesting trends. Other exanlples include webserver logs, streaI11S of transactional data froI11 large retail chains, and financial stock tickers.

When working with high-speed data strearlls, algoritlulls lllUSt be designed to construct data rnining rnodels while looking at the relevant data iterrlS *only once and in a .fixed order* (deternlined by the strearn-arrival pattern), with a lirnited arnount of main 111eI1l0ry. Data-strearn coruputatioll has given rise to several recent (theoretical and practical) studies of online or one-pass algorithrlls with bounded HIeIIIory. Algorithrns have been developed for one-pass cornputation of quantiles and order-statistics, estirnation of frequency I1l0Inents and join sizes, clustering and decision tree construction, estimating correlated aggregates, and cOInputing one-dirnensional (i.e., single-attribute) histogranls and 1Iaa1' wavelet decolllpositions. Next, we discuss one such algorithIn, for incremental rnaintenance of frequent itemsets.

26.7.1 Incremental Maintenance of Frequent Itemsets

Consider the Purchases Relation shown in Figure 26.1 and assurne that the minimum support threshold is 60%. It can be easily seen that the set of frequent iternsets of size 1 consists of *{pen }*, *{ink}*, and *{rnilk}* with supports of 100%, 75%, and 75%, respectively. T'he set of frequent itell1Sets of size 2 consists of *{pen, ink}* and *{pen, milk}*, both with supports of 75%. The Purchases relation is our first block of data. Our goal is to develop an algorithrIl that rnaintains the set of frequent itcrl1sets under insertion of new blocks of data.

As a first exarnple, let us consider the addition of the block of data shown in Figllre 26.13 to our original database (Figure 26.1). V'nder this addition, the set of frequent itcrIlsets does not change, although their support values do: *{pen}*, *{i'nk}*, and *{milk}* now have support values of 100%, 60%, and 60%, respectively, and *{pen, ink}* and *{pen, 'inilk}* now have 60% support. Note that we could detect this case of 'no change' sirnply by rnaintaining the nurnber of rnarket baskets in which each iternset occured. Irl this example, we update the (al)solute) support of itcrnset *{pen}* by 1.

transid	*custid*	*date*	*item*	*qty*
115	201	7/1/99	pen	2

Figure 26.13 The Purchases Relation Block 2

transid	*custid*	*date*	*item*	*qty*
115	201	7/1/99	water	1
115	201	7/1/99	lllilk	1

Figure 26.14 The Purchases Relation Block 2a

In general, the set of frequent itemsets Illay change. As an exalnple, consider the addition of the block shown in Figure 26.14 to the original database shown in Figure 26.1. We see a transaction containing the itern water, but we do not know the support of the iterllset $\{water\}$, since water was not above the IninInUlm support in our original database. A sirnple solution in this case is to rnake an additional scan over the original database and cornpute the support of the itenlset *{water}*. But can we do better? Another innnediate solution is to keep counters for *all* possible iterllsets, but the nUlnber of all possible itemsets is exponential in the nurnber of iterns-·-and most of these counters would be 0 anyway. Can we design an intelligent strategy that tells us *which* counters to ruaintain?

We introduce the notion of the **negative border** of a set of iternsets to help decide which counters to keep. The negative border of a set of frequent itemsets consists of all iterllsets *X* such that *X* itself is not frequent, but all subsets of X are frequent. For example, in the case of the database shown in Figure 26.1, the following iternsets rnake up the negative border: *{juice}*, *{water}*, and *{ink, milk}*. Now we can design a more efficient algorithm for maintaining frequent iternsets by keeping counters for all currently frequent iternsets *and* all iterllsets currently in the negative border. ()nly if an iternset in the negative border becomes frequent do we need to read the original dataset again, to find the support for new candidate itemsets that Blight be frequent.

We illustrate this point through the following t\vo exarnples. If we add Block 2a shown in Figure 26.14 to the original database shown in Figure 26.1, we increase the support of the frequent iterllset $\{milk\}$ by one, and we increase the support of the iternset *{water}*, which is in the negative border, by one as well. But since no iternset in the negative border beearne frequent, we do not have to re-scan the original database.

In eontrast, consider the addition of Block 2b shown in Figure 26.15 to the original database shown in Figure 26.1. In this case, the iternset *{juice}*, which was originally in the negative border, becornes frequent with a support of 60%. rrhis rneans that now the following itcrnsets of size two enter the negative border: $\{juice, pen\}$, *{juice, ink}*, and *{juice, milk}*. (We know that $\{juice, water\}$ cannot be frequent since the iteulset $\{water\}$ is not freqlient.)

[illegible]	[illegible]	[illegible]	[illegible]	[illegible]
115	201	7/1/99	juice	2
115	201	7/1/99	water	2

Figure 26.15 The Purchases Relation Block 2b

26.8 ADDITIONAL DATA MINING rfASKS

We focused on the problern of discovering patterns frorn a database, but there are several other equally inlportant data ruining tasks. We now discuss sollle of these briefly. The bibliographic references at the end of the chapter provide luauy pointers for further study.

- Dataset and Feature Selection: It is often irnportant to select the 'right' dataset to rnine. Dataset selection is the process of finding which datasets to uline. Feature selection is the proeess of deciding which attributes to include in the mining process.
- Sampling: One way to explore a large dataset is to obtain one or luore *samples* and analyze them. The advantage of sampling is that we can carry out detailed analysis on a sarnple that would be infeasible on the entire dataset, for very large datasets. The disadvantage of sampling is that obtaining a representative salllple for a given task is difficult; we rnight rniss irnportant trends or patterns because they are not reflected in the san1ple. Current database systerns also provide poor support for efficiently obtaining sanlples. Irnproving database support for obtaining sarnples with various desirable statistical properties is relatively straightforward and likely to be available in future DBMSs. Applying sarnpling for data ruining is an area for further research.
- Visualization: Visualization techniques can significantly assist in understanding cornplex datasets and detecting interesting patterns, and the importance of visualization in data ruining is widely recognized.

26.9 REVIEW QUESTIONS

Answers to the review questions can be found in the listed sections.

- What is the role of data rnining in theKI)I) process? (Secti.on 26.1)
- What is the a priori property? I)escribe an algorithnl for firlding frequent itcrIlsets. (Section 26.2.1)

- How are iceberg queries related to frequent iterIlsets? (Section 26.2.2)
- Give the definition of an *association rule*. What is the difference between support and confidence of a rule? (Setion 26.3.1)
- Can you explain extensions of association rules to ISA hierarchies? What other extensions of association rules are you farniliar with? (Sections 26.3.3 and 26.3.4)
- What is a sequential pattern? How can we cornpute sequential patterns? (Section 26.3.5)
- Can we use association rules for prediction? (Section 26.3.6)
- What is the difference bet\'leen Bayesian Networks and association rules? (Section 26.3.7)
- Can you give exanlples of classification and regression rules? How is support and confidence for such rules defined? (Section 26.3.8)
- What are the cOlnponents of a decision tree? How are decision trees constructed? (Sections 26.4.1 and 26.4.2)
- What is a cluster? What inforrnation do we usually output for a cluster? (Section 26.5)
- How can we define the distance between two sequences? Describe an algorithnl to find all sequences similar to a query sequence. (Section 26.6)
- Describe the block evolution Inodel and define the problclllS of increlnental rnodel maintenance and change detection. What is the added challenge in rnining data strearns? (Section 26.7)
- Describe an incrernental algorithnl for conlpllting frequent iternsets. (Section 26.7.1)
- Give exarnples of other tasks related to data rnining. (Section 26.8)

EXERCISES

Exercise 26.1 Briefly ans\ver the following questions:

1. Define *support* and *confidence* for an association 1"ule.
2. Expla.in why association rules cannot be used directly for prediction, \vithout further analysis or clornain knowledge.
3. What are the differences between *association rules*, *classification rules*, and *regression rules*?
4. \What is the difference between *classification* and *clustering*?

iru_ 1	*ustid*	*date*		*atu*
111	201	5/1/2002	ink	1
111	201	5/1/2002	lnilk	2
11.1	201	5/1/2002	.JuIce	1
112	105	6/3/2002	pen	1
112	105	6/3/2002	ink	1
112	105	6/3/2002	water	1
113	106	5/10/2002	pen	1
113	106	5/10/2002	water	2
113	106	5/10/2002	milk	1
114	201	6/1/2002	pen	2
114	201	6/1/2002	ink	2
114	201	6/1/2002	JUIce	4
114	201	6/1/2002	water	1
114	201	6/1/2002	ruilk	1

Figure 26.16 The Purchases2 Relation

5. What is the role of information visualization in data mining?
6. Give exarrlples of queries over a database of stock price quotes, stored as sequences, one per stock, that cannot be expressed in SQL.

Exercise 26.2 Consider the Purchases table shown in Figure 26.1.

1. Simulate the algorithrn for finding frequent iterllsets on the table in Figure 26.1 with *minsup=90* percent, and then find association rules with *m,inconJ=90* percent.
2. Can you modify the table so that the same frequent itemsets are obtained with *'fninsup=90* percent as with *minsup*=70 percent on the table shown in Figure 26.1?
3. Sirllulate the algorithrIl for finding frequent iternsets on the table in Figure 26.1 with *rn'insup=lO* percent and then find association rules with *rninconj=90* percent.
4. Can you modify the table so that the sarne frequent iternsets are obtained with *minsup*=10 percent as with *minsup*=70 percent on the table shown in Figure 26.1?

Exercise 26.3 Assulne we are given a dataset D of rnarket baskets and have computed the set of frequent iternsets $\mathcal{X}$ in 1) for a given support threshold *minsup*. Assume that we would like to add. another dataset D' to D, and rnaintain the set of frequent itmnsets with support threshold ***minsup*** in D U D'. Consider the following algorithrIl for incrernental Inaintenance of a set of frequent iternsets:

1. We run the ***a priori*** algoritlun on D' and find all frequent iterllsets in D' and their support. The result is a set of iterllsets $\mathcal{X}'$. We also cornpute the support of all itcrnsets X E $\mathcal{X}$ in *J)'*.
2. We then rnake a scan over *D* to cornpute the support of all iternsets in $\mathcal{X}'$.

Answer the following questions about the algorithm:

- The last step of the algorithm is rnissing; that is, what should the algorithm output'?
- Is this algorithm lllore efficient than the algorithm described in Section 26.7.1'1

Exercise 26.4 Consider the Purchases2 table shown in Figure 26.16.

- List all iterllsets in the negative border of the dataset.
- List all frequent itelnsets for a support threshold of 50%.
- Give an exaruple of a database in which the addition of this database does not change the negative border.
- Give an exarnple of a database in which the addition of this database would change the negative border.

Exercise 26.5 Consider the Purchases table shown in Figure 26.1. Find all (generalized) association rules that indicate the likelihood of items being purchased on the same date by the same customer, with *minsup* set to 10% and *minconj* set to 70%.

Exercise 26.6 Let us develop a new algorithm for the computation of all large itemsets. Assume that we are given a relation D siInilar to the Purchases table shown in Figure 26.1. We partition the table horizontally into k parts $D_1, \dots, D_k$.

1. Show that, if itemset X is frequent in D, then it is frequent in at least one of the k parts.
2. Use this observation to develop an algorithm that cornputes all frequent itemsets in two scans over D. (Hint: In the first scan, compute the locally frequent itemsets for each part D_i, i E {I, ... , k}.)
3. Illustrate your algorithm using the Purchases table shown in Figure 26.1. The first partition consists of the two transactions with *transid* 111 and 112, the second partition consists of the two transactions with *transid* 113 and 114. Assulne that the minimum support is 70 percent.

Exercise 26.7 Consider the Purchases table shown in Figure 26.1. Find all sequential patterns with *minsup* set to 60%. (The text only sketches the algorithm for discovering sequential patterns, so use brute force or read one of the references for a complete algorithm.)

Exercise 26.8 Consider the SubscriberInfo Relation shown in Figure 26.17. It contains information about the marketing cmnpaign of the *DB Aficionado* magazine. The first two colurnns show the age and salary of a potential customer and the *subscription* colurnn shows whether the person subscribes to the rnagazine. \Ve want to use this data to construct a decision tree that helps predict whether a person will subscribe to the 11lagazine.

1. Construct the AVC-group of the root node of the tree.
2. Assume that the spliting predicate at the root node is $age \leq 50$. Construct the AVC-groups of the two children nodes of the root node.

Exercise 26.9 Assurne you are given the following set of six records: (7,55), (21, 202), (25,220), $\langle 12, 73\rangle$, $\langle 8, 61\rangle$, and $\langle 22, 249\rangle$.

1. Assurning that all six records belong to a single cluster, cornpute its center and radius.
2. Assurne that the first three records belong to one cluster and the second three records belong to a different cluster. COlnpute the center and radius of the two clusters.
3. \Vhich of the two clusterings is 'better' in your opinion and why?

Exercise 26.10 Asslune you are given the three sequences (1, 3, 4), $\langle 2, 3, 2\rangle$, $\langle 3, 3, 7\rangle$. COlnpute the Euclidian Bonn between all pairs of sequences.

age	*salary*	*subscription*
37	45k	No
39	70k	Yes
56	50k	Yes
52	43k	Yes
35	90k	Yes
32	54k	No
40	58k	No
55	85k	Yes
43	68k	Yes

Figure 26.17 The SubscriberInfo Relation

BIBLIOGRAPHIC NOTES

Discovering useful knowledge from a large database is lllore than just applying a collection of data rnining algorithms, and the point of view that it is an iterative process guided by an analyst is stressed in [265] and [666]. Work on exploratory data analysis in statistics, for example [745], and on rnachine learning and knowledge discovery in artificial intelligence was a precursor to the current focus on data lTlining; the added ernphasis on large volunles of data is the inlportant new elernent. Good recent surveys of data mining algorithms include [267, 397, 507]. [266] contains additional surveys and articles on many aspects of data mining and knowledge discovery, including a tutorial on Bayesian networks [:371]. The book by Piatetsky-Shapiro and Frawley [595] contains an interesting collection of data rnining papers. The annual SIGKDD conference, run by the ACM special interest group in knowledge discovery in databases, is a good resource for readers interested in current research in data mining (25, 162, 268, 372, 613, 691], as is the *Journal of Knowledge D'iscovery and Data Mining.* [363, 370, 511, 781] are good, in-depth textbooks on data nlining.

The problern of mining association rules was introduced by Agrawal, Itnielinski, and Swami [20]. l\!1any efficient algorithnls have been proposed for the cornputation of large iternsets, including [21,117,364,683,738,786].

Iceberg queries have been introduced by F'ang et al. [264]. There is also a large body of research on generalized forrns of <lssociation rules; for example, [700, 701, 703]. The problem of finding rnaxirnal frequent itelnsets has also received significant attention [13, 67, 126, 346, 347, 479, 787]. Algorithrns for rnining association rules with constraints are considered in [68,462, 563, 590, 591, 703].

Parallel algorithnls are described in [23] and [655]. Recent papers on parallel data ruining can be found in [788], and work on distributed data 111ining can be found in [417].

[291] presents an aigoritlull for discovering association rules over a continuous nUllwric attribute; association rules over numeric attributes are also discussed in [78:3J. The general fornl of association rules, in which attributes other than the transaction id are grouped is developed in [529]. Association rules over iterns in a hierarchy are discllssed in [361, 700]. Further

extensions and generalization of association rules are proposed in [67, 115, 563]. Integration of rnining for frequent itemsets into database systcrns has been addressed in [654, 743]. The problern of Inining sequential patterns is discussed in [24], and further algorithrIls for rnining sequential patterns can be found in [510, 702].

General introductions to classification and regression rules can be found in [362, 532]. The classic reference for decision and regression tree construction is the CART book by Breilnan, Friedmau, Olsheu, and Stone [111]. A lnachine learning perspective of decision tree construction is given by Quinlan [603]. Recently, several scalable algorithnls for decision tree construction have been developed [309, 311, 521, 619, 674J.

'rhe clustering problern has been studied for decades in several disciplines. Sample textbooks include [232, 407, 418]. Scalable clustering algorithuls include CLARANS [562], DBSCAN [249, 250], BIRCH [798], and CURE [344]. Bradley, Fayyad, and Reina address the problem of scaling the K-Means clustering algorithm to large databases [108, 109]. The problern of finding clusters in subsets of the fields is addressed in [19]. Ganti et al. exauline the problerll of clustering data in arbitrary rnetric spaces [302]. Algorithrlls for clustering caterogical data include STIRR [315J and CACTUS [301]. [651] is a clustering algorithm for spatial data.

Finding siulilar sequences from a large database of sequences is discussed in [22, 262, 446, 606,680].

Work on incrernental rnaintenance of association rules is considered in [174, 175, 736]. Ester et al. describe how to nlaintain clusters incrernentally [248], and Hidber describes how to rnaintain large iteulsets incrernentally [378]. There has also been recent work on rnining data strearns, such as the construction of decision trees over data streams [228, 309, 393] and clustering data streanlS [343,568]. A general framework for ruining evolving data is presented in [299]. A framework for measuring change in data characteristics is proposed in [300J.

27

INFORMATION RETRIEVAL ANDXMLDATA

- How are DBMSs evolving in response to the growing aIllounts of text data?
- What is the vector space rnodel and how does it support text search?
- How are text collections indexed?
- Cornpared to IR systenls, what is new in Web search?
- How is XML data different from plain text and relational tables?
- What are the main features of XQuery?
- What are the irnplementation challenges posed by XML data?
- Key concepts: information retrieval, boolean and ranked queries; relevance, precision, recall; vector space model, TF/IDF terrn weighting, document similarity; inverted index, signature file; Web crawler, hubs and authorities, Pigeon Rank of a webpage; sernistructured data lllodel, XML; XQuery, path expressions, FLWR queries; XML storage and indexing

'with Raghav Kaushik
University of Wisconsin-Madison

A *memex* is a device in which an individual stores all his books, records, and cornrnunications, and which is rnechanized so that it rnay be consulted with exceeding speed and flexibility.

—Vannevar Bush, *As We May Think, 1945*

The field of inforlnation retrieval (IR) has studied the problenl of sea,rching collections of text docurnents since the 19508 and developed largely independently of database systenls. The proliferation of text docunlents on the Web lllade docurnent search an everyday operation for 1110st people and led to renewed research on the topic.

The database field's desire to expand the kinds of data that can be managed in a DBMS is well-established and reflected in developments like object-relational extensions (Chapter 23). Documents on the Web represent one of the rnost rapidly growing sources of data, and the challenge of rnanaging such documents in a DBMS has naturally become a focal point for database research.

The Web, therefore, brought the two fields of database rnanagement systenls and information retrieval closer together than ever before, and, as we will see, XML sits squarely in the middle ground between thenl. We introduce IR systems as well as a data model and query language for XML data and discuss the relationship with (object-)relational database systerns.

In this chapter, we present an overview of information retrieval, Web search, and the emerging XML data model and query language standards. We begin in Section 27.1 with a discussion of how these text-oriented trends fit within the context of current object-relational database systeIns. We introduce inforrnation retrieval concepts in Section 27.2 and discuss specialized indexing techniques for text in Section 27.3. We discuss Web search engines in Section 27.4. In Section 27.5, we briefly outline current trends in extending database systems to support text data and identify SOllle of the irnportant issues involved. In Section 27.6, we present the XML data lllodel, building on the XML concepts introduced in Chapter 7. We describe the XQuery language in Section 27.7. In Section 27.8, we consider efficient evaluation of XQuery queries.

27.1 COLLIDING WORLDS: DATABASES, IR, AND XML

'The *\i\Teb* is the rnost widely used doculnent collection today, and search on the Web differs froIn traditional IR-style docurnent retrieval in iluportant ways. First, there is great emphasis on scalability to very large document collections. IR systerns typically dealt with tens of thousands of documents, whereas the Web contains billions of pages.

Second, the Web has significantly changed how docurnent collections are created and used. Traditionally, IR systerlls were aimed at professionals like librarians and legal researchers, who were trained in using sophisticated retrieval engines. Docurnents were carefully prepared, and docllrnents in a given collection were typically on related topics. On thevVeb, docurnents are created by an infinite variety of individuals for equally many purposes, and reflect this diversity in size and content. Searches are carried out by ordinary people with no training in using retrieval software.

The ernergence of XML has added a third interesting diInensioI1 to text search: Every cloClunent can no\v be rnarked up to reflect additional infol'lnation of interest, such as authorship, source, and even details about the intrinsic content. This has changed the nature of a “document” froIn free text to textual objects with associated fields containing metadata (data about data) or descriptive infonnation. Links to other docurnents are a particularly irnportant kind of lnetadata, and they can have great value in searching docurnent collections on the Web.

The Web also changed the notion of what constitutes a docunlent. Documents on the Web may be multinledia objects such as irnages or video clips, with text appearing only in descriptive tags. We must be able to Inanage such heterogeneous data collections and support searches over thern.

Database rnanagernent systenls traditionally dealt with simple tabular data. In recent years, object-relational database systerns (ORDBMSs) were designed to support complex data types. Images, videos, and textual objects have been explicitly rnentioned as exaruples of the data types ORDBMSs are intended to support. Nonetheless, current database systerns have a long way to go before they can support such cOlnplex data types satisfactorily. In the context of text and XML data, challenges include efficient support for searches over textual content and support for searches that exploit the loose structure of XML data.

27.1.1 DBMS versus IR Systems

Database and IR systcrns have the COllllnon objective of supporting searches over collections of data. However, rnany irnportant differences have influenced their developrnent.

- Searches versus Queries: IR systerns are designed to support a specialized class of queries that we also call searches. Searches are specified in ternlS of a few search terms, and the underlying data is usually a collection of unstructured text docurnents. III addition, an irnportant feature of IR searches is that search resultsrnay be ranked, or ordered, in tcrrns of how ‘well’ the search results rnatch the search terms. In contrast, database systerns support a very general class of queries, and the underlying data is rigidly structured. Unlike III systems, database systerns have traditionally returnedunranked sets of results. (Even the recent SQL/OLAP extensions that support early results and searches over ordered data (see Chapter 25) do not order results in terlTIS of how well they rnatch the query. Relational queries are *precise* in that a ro\v is either in the answer or it is not ; there is no notion of 'how well a row matches' the query.) In other words, a relational query only assigns two ranks to a row, indicating 'whether the row is in the ans\ver or not.

- Updates and Tr'ansactions: IR systelns are optirnized for a read-Illostly workload and do not support the notion of a transaction. In traditional IR systerlls, ne\v docurnents are added to the doculnent collection frorH tirne to time, and index structures that speed up searches are periodically rebuilt or updated. Therefore, docllrnents that are highly relevant for a search rnight exist in the IR systeln, but not be retrievable yet because of outdated index structures. In contrast, database systerns are designed to handle a wide range of workloads, including update-intensive transaction processing workloads.

These differences in design objectives have led, not surprisingly, to very different research elnphases and system designs. Ilesearch in IR studied ranking functions extensively. For example, arllong other topics, research in IR investigated how to incorporate feedback frOITl a user's behavior to modify a ranking function and how to apply linguistic processing techniques to improve searches. Database research concentrated on query processing, concurrency control and recovery, and other topics, as covered in this book.

The differences between a DBMS and an IR systenl from a design and irnplementation standpoint should become clear as we introduce IR systerlls in the next few sections.

27.2 INTRODUCTION TO INFORMATION RETRIEVAL

There are two COrllr110n types of searches, or queries, over text collections: boolean queries and ranked queries. In a boolean **query**, the user specifies an expression constructed using terlllS and boolean operators (And, Or, Not). For exalnple,

database And *(lvlicT08ojt* Or *IBM)*

This query asks for all docurnents that contain the terrn *database* and in addition, either *Microsoft* or *IBM.*

In a ranked query the user specifies one or rnore terrlls, and the result of the query is a list of docurllents ranked by their relevance to the query. Intuitively, docurnents at the top of the result list are expected to 'rnatch' the search

docid	Document
1	agent Janles Bond good agent
2	agent rnobile cornputer
3	James Madison Inovie
4	Janles Bond movie

Figure 27.1 A Text Database with Four Records

condition ruore closely, or be 'rnore relevant', than doculnents lower in the result list. While a document that contains *Microsoft* satisfies the search'*Microsoft, IBM,'* a document that also contains *IBM* is considered to be a better match. Similarly, a doculnent that contains several occurrences of *Microsoft* might be a better rnatch than a document that contains a single occurence. Ranking the docurnents that satisfy the boolean search condition is an important aspect of an IR search engine, and we discuss how this is done in Sections 27.2.3 and 27.4.2.

An important extension of ranked queries is to ask for documents that are most relevant to a given natural language sentence. Since a sentence has linguistic structure (e.g., subject-verb-object relationships), it provides more information than just the list of words that it contains. We do not discuss natural language search.

27.2.1 Vector Space Model

We now describe a widely-used franlework for representing docurnents and searching over docurnent collections. Consider the set of all terrns that appear in a given collection of documents. We can represent each document as a vector with one entry per ternl. In the shnplest 1'01'111 of doclunent vectors, if terrn *j* appears *k* tirnes in dOCUlnent i, the document vector for docurnent i contains value *k* in position *j*. The docurnent vector for *i* contains the value 0 in positions corresponding to terrns that do not appear in i.

Consider the exaInple collection of four docurnents shown in Figure 27.1. rrhe docUluent vector representation is illustrated in Figure 27.2; each row represents a docurnent. This representation of docurnents as terrn vectors is called the vector space model.

docid	agent	Bond	computer	good	James	Madison	mobile	movie
1	2	1	0	1	1	0	0	0
2	1	0	1	0	0	0	1	0
3	0	0	0	0	1	1	0	1
4	0	1	0	0	1	0	0	1

Figure 27.2 Document Vectors for the Example Collection

27.2.2 TFIIDF Weighting of Terms

We described the value for a terril in a document vector as simply the term frequency (TF), or nurnber of occurrences of that terrn in the given document. This reflects the intuition that a term which appears often is more iInportant in characterizing the document than a terrn that appears only once (or a term that does not appear at all).

However, some terms appear very frequently in the document collection, and others are relatively rare. The frequency of terms is eITlpirically observed to follow a Zipfian distribution, as illustrated in Figure 27.3. In this figure, each position on the X-axis corresponds to a terrll and the Y-axis corresponds to the nUlnber of occurrences of the term. Terms are arranged on the X-axis in decreasing order by the nurnber of tirnes they occur (in the docurnent collection as a whole).

As rnight be expected, it turns out that extremely COmlTIOn terms are not very useful in searches. Examples of such common terms include *a, an, the* etc. Terrns that occur extremely often are called stop words, and docunlents are pre-processed to elirilinate stop words.

Even after eliminating stop words, we have the phenorilenon that some words appear nluch luore often than others in the docurnent collection. Consider the words *Linux* and *kernel* in the context of a collection of dOCUlnents about the Linux operating systern. While neither is COlnrnon enough to be a stop word, *Linux* is likely to appear much rnore often. Given a search that contains both these keywords, we are likely to get better results if we give Inore irnportance to docurnents that contain *kernel* than docurnents that contain *Linux.*

We can capture this intuition by refining the docurnent vector representatioll as follows. The value associated with ternl j in the docurnent vector for docurnent i, denoted as w_{ij}, is obtained by rnultiplying the terrIl frequency t_{ij} (the nuruber of tirnes term j appears in docurnent i) by the inverse docurnent frequency (IDF) of terrn j in the docurnent collection. IDF of a tenn j is defined as *log(lVInj);* where N is the total rHnnber of dOCUlnents, and n_j is the nurnber of cloClllnents that tenn *j* appears in. This effectively increases the weight given to rare tenns. As an example, in a collection of 10,000 docurnents, a terrIl that appears in half the docurnents has an IDF of 0.3, and a tenll that occurs in just one docurnent has an IDF of 4.

Length Normalization

Consider a docurnent *!J.* Suppose that we lnodify it by adding a large nUlllber of new terrns. Should a the weight of a terrn *t* that appears in *D* be the saIne in the doclunent vectors for *D* and the rnodified dOCUlTlent? Although the TFjIDF weight for t is indeed the saIne in the two document vector, our intuition suggests that the weight should be less in the 1110dified document. Longer docul'llents tend to have lnore terms, and lnore occurrences of any given terrn. Thus, if two doculnents contain the saIne nUlnber of occurrences of a given tenll, the importance of the ten'll in characterizing the document also depends on the length of the docull1ent.

Several approaches to length nornlalization have been proposed. Intuitively, all of ther'll reduce the irnportance given to how often a term occurs as the frequency grows. In traditional IR systelns, a popular way to refine the sirnilarity Inetric is cosine length normalization:

$$w^*_{ij} = \frac{w_{ij}}{\sqrt{\sum_{k=1}^{t} w_{ik}^2}}$$

In this formula, t is the nurnbei' of tenns in the dOCulnent collection, w_{ij} is the TFjIDF weight without length norrnalization, and w^*_{ij} is the length adjusted TFjlDF weight.

Tenns that occur frequently in a doculnent are particularly problenlatic on the Web because webpages are often deliberately rnodified by adding rnany copies of certain words···· for exarnple, sale, free, sex to increase the likelihood of their being returned in response to queries. For this reason, Web search engines typically norrnalize for length by imposing a lnaxirnurn value (usually 2 or 3) for terrIl frequencies.

27.2.3 Ranking Document Similarity

We now consider how the vector space representation allows us to rank docu-rnents in the result of a ranked query. A key observation is that a ranked query can itself be thought of as a docUlllent, since it is just a collection of terrl1s. 1'his allows us to use document similarity as the basis for ranking query results—the doculnent that is rnost sirnilar to the query is ranked highest, and the one that is least sirnilar is ranked lowest.

If a total of t teru18 appear in the collection of docurnents (t is 8 in the exaulple shown in Figure 27.2), we can visualize document vectors in a t-dilnensional space in \vhich each axis is labeled with a tel'1n. This is illustrated in Figure 27.4, for a two-dirnensional space. The figure shows doculuent vectors for two documents, D_1 and D_2, as well as a query Q.

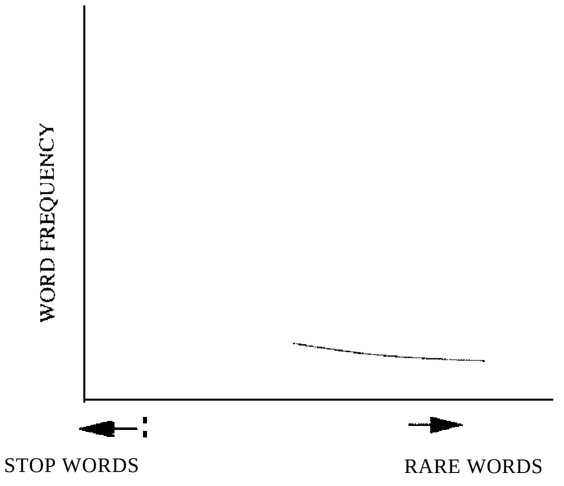

Figure 27.3 Zipfian Distribution of Term Frequencies

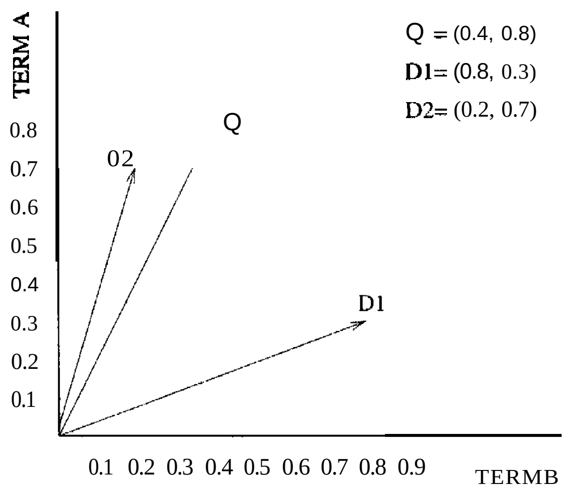

Figure **27.4** Document Similarity

The traditional rneasure of closeness between two vectors, their *dot product*, is used as a lneasure of docurnent silnilarity. The silnilarity of query Q to a doculnent D_i is Illea8Ured by their dot produet:

$$sim(Q, D_i) - \sum_{.1=1}^{t} q_j^*.w_{ij}^*$$

In the example shown in Figure 27.4, $sim(Q, D_1)$ (0.4 * 0.8) .+ (0.8 * 0.3) = 0.56, and $siln(Q, D2)$ = (0.4 *0.2)-+- (0.8 *0.7) = 0.64. Accordingly, $D2$ is ranked higher than 1)1 in the search result.

In the context of the Web, docurnent sirnilal'ity is one of several IneaSU1'es that can be used. to rank results, but should not be used exclusively. First, it is questionable whether users want dOCllrnents that are sirnilar to the query (which typically consists of one or two 'words) or dOCUlI.lenS that contain useful inforrnation related to the quer,Y tel'111S. IntuitivelY,we want to give ilnportance to the *quality* of a Web page while ranking it, in addition to reflecting the sirnilarity of the page to a given query. Links between pages provide valuable additional infornnation that can be used to obtain high-quality results. We discuss this issue in Section 27.4.2.

27.2.4 Measuring Success: Precision and Recall

Two criteria are cornnlonly used to evaluate information retrieval systerlls. Precision is the percentage of retrieved documents that are relevant to the query. Recall is the percentage of relevant docurnents in the database that are retrieved in response to a query.

Retrieving all documents in response to a query trivially guarantees perfect recall, but results in very poor precision. The challenge is to achieve good recall together with high precision.

In the context of search over the Web, the size of the underlying collection is on the order of billions of docuruents. Given this, it is questionable whether the traditional measure of recall is very useful. Since users typically don't look beyond the first screen of results, the quality of a Web search engine is largely deterlnined by the results shown on the first page. The following adapted definitions of precision and recall rnight be more appropriate for Web search engInes:

- Web Search Precision: The percentage of results on the first page that are relevant to the query.
- Web Search Recall: rrhe fraction N/M, expressed as a percentage, where M is the nUluber of results displayed on the front page, and of the M ruost relevant documents, N is the number displayed on the front page.

27.3 INDEXING FOR TEXT SEARCH

In this section, we introduce two indexing techniques that support the evaluation of boolean and ranked queries. 'The '*inverted index* structure discussed in Section 27.3.1 is widely used due to its sirnplicity and good perforlnance. Its rnain disadvantage is that it imposes a significant space overhead: The size can be up to 300 percent the size of the original file. The *signature file* index discussed in Section 27.3.2 has a sInall space overhead and offers a quick filter that elirninates rnost nonqualifying docurnents. However, does not scale as well to larger datahase sizes because the index has to be sequentially scanned.

Before a doeuruent is indexed, it is typically pre-processed to elirninate stop words. Since the size of the indexes is very sensitive to the nurnber of tern1S in the docurnent collection, elirninating stop words can greatly reduce index size. IR systcrns also do certain other kinds of pre-processing. :For instance, they apply stelnming to reduce related terrns to a ca,nonical forrn. This step also reduces the nUluber of terrn8 to be indexed, but equally irnportantly, it allows us to retrieve documents that lnay not contain the exact query terrIl but contain s(Hne variant. As an example, the terrns *run, running,* and *runner* all stern to *run.* The terrIl *run* is indexed, and every occurrence of a variant of this term is treated as an occurrence of *run.* A query that specifies *runner* finds docurnents that contain any word that stenlS to *run.*

27.3.1 Inverted Indexes

An inverted index is a data structure that enables fast retrieval of all documents that contain a query terr11. For each ternl, the index rnaintains a list (called the inverted list) of entries describing occurrences of the tenn, with one entry per docurnent that contains the ternl.

Consider the inverted index for our running example shown in Figure 27.5. The term 'Jarnes' has an inverted list with one entry each for documents 1, 3, and 4; the term 'agent' has entries for docurnents 1 and 2.

The entry for document *d* in the inverted list for terrn t contains details about the occurrences of term *t* in document *d.* In Figure 27.5, this information consists of a list of locations within the document that contain term *t.* Thus, the entry for document 1 in the inverted list for terrn 'agent' lists the locations 1 and 5, since 'agent' is the first and fifth word of docurnent 1. In general, we can store additional information about each occurrence (e.g., in an HTML docurnent, is the occurrence in the TITLE tag?) in the inverted list. We can also store the length of the docurnent if this is used for length norlnalization (see below).

The collection of inverted lists is called the postings file. Inverted lists can be very large for large doeurnent collections. In fact, Web search engines typically store each inverted list on a separate page, and Inost lists span rnultiple pages (and if so, are rnaintained as a linked list of pages). In order to quickly find the inverted list for a, query terrn, all possible query terrns are organized in a second index structure such as a B+ tree or a hash index.

The second index, called the lexicon, is Inuch srnaller than the postings file since it only contains one entry per terrn, and further, only contains entries for the set of terl11S that are retained after elirninating stop words, and applying stenlluing rules. An entry consists of the terlIl, sOH1C surnrnary inforrnation about its inverted list, and the address (on disk) of the inverted list. In Figure 27.5, the SllIl1lnary inforrnation consists of the llllllber of entries in the inverted

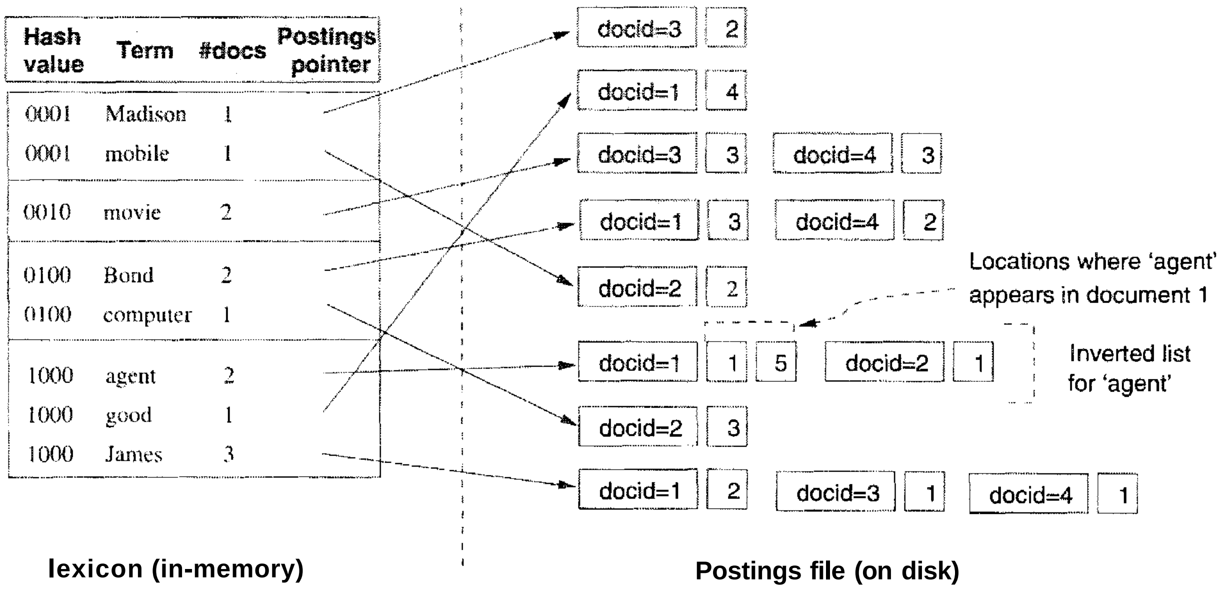

Figure 27.5 Inverted Index for Example Collection

list (i.e., the nurnber of documents that the terl11 appears in). In general, it could contain additional infonnation such as the IDF for the terrIl, but it is il11portant to keep the entry's size as slllall as possible.

The lexicon is rua.intained in-lllelnory, and enables fast retrieval of the inverted list for a query terrn. rrhe lexicon in Figure 27.5 uses a hash index, and is sketched by showing the hash value for the terrn; entries for terms are grouped into hash buckets by their hash value.

Using an Inverted Index

A_ query containing a single tenn is evaluated by first searching the lexicon to find the address of the inverted list for the terrIl. Then the inverted list is retrieved, the docids in it are rnapped to physical doculnent addresses, and the corresponding docurnents are retrieved. If the results are to be ranked, the relevance of each docurnent in the inverted list to the query term is C01l1puted, and docurnents are then retrieved in order of their relevance rank. ()bserve that the inforrna,tion needed to cornpute the relevance measure described in Section 27.2 --the frequency of the query ternl in the dOCu1nent, the IDF of the terrn in the docurnent collection, and the length of the docurnent if it is used for length nonnalizatioll-------are all available in either the lexicon or the inverted list.

When inverted lists are very long, as in Web search engines, it is useful to consider \vhether we should precornpute the relevance of each dOCUlnent in the inverted list for a terrn (with respect to that terrn) and sort the list by relevance rather than docurnent id. This would speed up querying because we can just look at a prefix of the inverted list, since users rarely look at 11101'0 than the first few results. However1 maintaining lists in sorted order by relevance can be expensive. (Sorting by dOcUll1cnt id is convenient because new dOCUlnents are assigned increasing ids, and we can therefore sirnply append entries for new dOCUlnents at the end of the inverted list. Further, if the sirnilarity function is changed, we do not have to rebuild the index.)

A query with a conjunction of several terrns is evaluated by retrieving the inverted lists of the query terrns one at a time and intersecting theln. In order to rninirnize memory usage, the inverted lists should be retrieved in order of increasing length. A query with a disjunction of several terrns is evaluated by ruerging all relevant inverted lists.

Consider the exaruple inverted index shown in Figure 27.5. To evaluate the query 'JaUles', we probe the lexicon to find the address of the inverted list for 'Ja1nes', fetch it from disk and then retrieve docurIlent 1. To evaluate the query ,Jarnes' AND 'Bond', we first retrieve the inverted list for the tenn 'Bond' and intersect it with the inverted list for the terrn 'Janles.' (The inverted list of the terrn 'Bond' has length two, whereas the inverted list of the terrIl 'Jarnes' has length three.) rrhe result of the intersection of the list (1,4) with the list (1, 3, 4) is the list (1,4) and doculuents 1 and 4 are therefore retrieved. 'fa evaluate the query '.lalnes' OR 'Bond,' we retrieve the two inverted lists in any order and merge the results.

For ranked queries with lnultiple tenns, we 1nust fetch the inverted lists for all terrl1s, COlllpute the relevance of every doclunent that appears in one of these lists with respect to the given collection of query terrns, and then sort the docurnent ids by their relevance before fetching the docluuents in relevance rank order. Again, if the inverted lists are sorted by the relevance measure, we can support ranked queries by typically processing only sluall prefixes of the the inverted lists. (()bserve that the relevance of a docuIllent with respect to the query is easily cornputed froIn its relevance with respect to each query term.)

27.3.2 Signature Files

A **signature file** is another index structure for text database systerns that supports efficient evaluation of boolean queries. A signature file contains an index record for each docurnent in the database. This index record is called the **signature** of the dOClunent. Each signature has a fixed size of b bits; b is called the **signature width.** rrhe bits that are set depend on the words that appear in the docllrnent. We rnap words to bits by applying a hash function to ea,ch word in the docurnent and we set the bits that appear in the result of

[illegible]	[illegible]	[illegible]
1	agent Jarnes Bond good agent	1100
2	agent mobile C0111puter	1101
3	James Madison lnovie	1011
4	JaInes Bond rnovie	1110

Figure 27.6 Signature File for Example Collection

the hash function. Note that unless we have a bit for each possible word in the vocabulary, the same bit could be set twice by different words because the hash function maps both words to the saIne bit. We say that a signature S_1 matches another signature 82 if all the bits that are set in signature 8_2 are also set in signature 8_1. If signature 8_1 Inatches signature 8_2, then signature 8_1 has at least as many bits set as signature 8_2.

For a query consisting of a conjunction of terms, we first generate the query signature by applying the hash function to each word in the query. We then scan the signature file and retrieve all documents whose signatures match the query signature, because every such document is a potential result to the query. Since the signature does not uniquely identify the words that a docuInent contains, we have to retrieve each potential rnatch and check whether the doculnent actually contains the query terms. A documnent whose signature matches the query signature but that does not contain all terms in the query is called a false positive. A false positive is an expensive rnistake since the documnent has to be retrieved froln disk, parsed, stemrned, and checked to determine whether it contains the query terms.

For a query consisting of a disjunction of tenns, we generate a list of query signatures, one for each terrn in the query. The query is evaluated by scanning the signature file to find documnents whose signatures rnatch any signature in the list of query signatures.

As an exarllple, consider the signature file of width 4 for our running exarnple shown in Figure 27.6. rrhe bits set by the hashed values of all query terrns are shown in the figure. To evaluate the query 'Jal1les,' we first cOlnpute the hash value of the terrn; this is 1000. Then we scan the signature file and find rnatching index records. As we can see fronl Figure 27.6, the signatures of all records have the first bit set. We retrieve all doculnents and check for false positives; the only false positive for this query is documnent with rid 2. (lJnfortunately, the hashed value of the terrn 'agent' also happened to set the very first bit in the signature.) Consider the query 'James' And 'Bond.' The query signature is llOO and three documnent signatures rnatch the query signature. Again, \ve retrieve one false positive. As another exarnple of a conjunctive query, consider the query 'movie' And 'Madison.' The query signature is 0011, and only one doclunent signature lnatches the query signature.No false positives are retrieved.

Note that for each query we have to scan the cOlllplete signature file, and there are as 11lany records in the signature file as there are documents in the database. To reduce the anlount of data that has to be retrieved for each query, we can vertically partition a signature file into a set of bit slices, and we call such an index a bit-sliced signature file. The length of each bit slice is still equal to the number of doculllents in the database, but for a query with q bits set in the query signature we need only to retrieve q bit slices. The reader is invited to construct a bit-sliced signature file and to evaluate the exarnple queries in this paragraph using the bit slices.

27.4 WEB SEARCH ENGINES

Web search engines rllust contend with extreruely large nurubers of doculllents, and have to be highly scalable. Docurnents are also linked to each other, and this link infonnation turns out to be very valuable in finding pages relevant to a given search. These factors have caused search engines to differ frorn traditional IR systerns in irnportant ways. Nonetheless, they rely on sorne forn1 of inverted indexes as the basic indexing mechanism. In this section, we discuss Web search engines, using Google as a typical example.

27.4.1 Search Engine Architecture

Web search engines crawl the web to collect docurnents to index. 'Ihe crawling algorithrn is sirrlple, but crawler software can be cornplex because of the details of connecting to millions of sites, minimizing network latencies, parallelizing the crawling, dealing with tirneouts and other connection failures, ensuring that crawled sites are not unduly stressed by the cra\vler, and other practical concerns.

The search algorithrn used by a crawler is a graph traversal. Starting at a collection of pages with rnany links (e.g.,Yahoo directory pages), all links on cra\vled pages are followed to identify new pages. This step is iterated, keeping track of which pages have been visited in order to avoid re-visiting thenl.

The collection of pages retrieved through crawling can be enonnous, on the order of billions of pages. Indexing thern is a very expensive task. Fortunately, tlle task is highly parallelizable: Each docurnent is independently arlalyzed to create inverted lists for the terrns that appear in the docurnent. These per-doCUlnent lists are then sorted by terrn and luerged to create cornplete per-term inverted lists that span all dOCllrnents. Ternl statistics such as IDF can be cornputed during the lnerge phase.

Supporting searches over such vast indexes is another luanulloth undertaking. Fortunately, again, the task is readily parallelized using a cluster of inexpensive Inachines: We can deal with the anlount of data by partitioning the index across several rnachines. Each Inachine contains the inverted index for those terms that are Inapped to that luachine (e.g., by hashing the tenn). Queries 111ay have to be sent to luultiple Inachines if the terrl1S they contain are handled by different rnachines, but given that Web queries rarely contain rnore than two terrns, this is not a serious probleln in practice.

We rnust also deal \vith a huge volume of queries; Google supports over 150 lllillion searches each day, and the nUlnber is growing. This is acc(nnplished by replicating the data across several machines. We already described how the data is partitioned across Inachines. For each partition, we now assign several nlachines, each of which contains an exact copy of the data for that partition. Queries on this partition can be handled by any rnachine in the partition. Queries can be distributed across rnachines on the basis of load, by hashing on IP addresses, etc. Replication also addresses the problern of high-availability, since the failure of a Inachine only increases the load on the remaining rnachines in the partition, and if partitions contain several rnachines the ilnpact is sIuall. Failures can be rnade transparent to users by routing queries to other Inachines through the load balancer.

27.4.2 Using Link Information

webpages are created by a variety of users for a variety of purposes, and their content does not always lend itself to effective retrieval. The rnost relevant pages for a search rnay not contain the search terrns at all and are therefore not returned by a boolean keyword search! For exarnple, consider the query ternl 'Web browser.' A boolean text query using the tenns does not return the relevant pages of Netscape Corporation or Microsoft, because these pages do not contain the terrn 'Web browser' at all. Sirnilarly, the horne page of 'Yahoo does not contain the terrn 'search engine.' The problenl is that relevant sites do not necessarily describe their contents in a way that is llseful for boolean text queries.

Until now, we only considered infonnation 'within a single \vebpage to estirnate its relevance to a query. But webpages are connected through h:yperlinks, and it is quite likely that there is a webpage containing the terrn 'search engine' that has a link to Yahoo's horne page. Can we use the infornation hidden in such links'?

Building on research in the sociology literature, an interesting analogy between links and bibliographic citations suggests a \vay to exploit link infoI'Ination: Just as influential authors and pubications are cited often, good webpages are likely to be often linked to. It is useful to distinguish between two types of pages, *authorities* and *hubs*. An authority is a page that is very relevant to a certain topic and that is recognized by other pages as authoritative on the subject. These other pages, called hubs, usually have a significant nUll1ber of hyperlinks to authorities, although they themselves are not very well known and do not necessarily carry a lot of content relevant to the given query. Hub pages could be cOlnpilatiol1s of resources about a topic on a site for professionals, lists of reco111mended sites for the hobbies of an individual user, or even a part of the bookIllarks of an individual user that are relevant to one of the user's interests; their Blain property is that they have IHany outgoing links to relevant pages. Good hub pages are often not well known and there may be few links pointing to a good hub. In contrast, good authorities are 'endorsed' by rnany good hubs and thus have many links froln good hub pages.

This symbiotic relationship between hubs and authorities is the basis for the HITS algoritlun, a link-based search algorithm that discovers high-quality pages that are relevant to a user's query terrns. The HITS algorithIll rnodels Web as a directed graph. Each webpage represents a node in the graph, and a hyperlink froIn page A to page B is represented as an edge between the two corresponding nodes.

Assulne that we are given a user query with several terIns. The algorithIll proceeds in two steps. In the first step, the *sarnpling step*, we collect a set of pages called the base set. The base set 11lOSt likely includes very relevant pages to the user's query, but the base set can still be quite large. In the second step, the *iteration step,* we find good authorities and good hubs arnong the pages in the base set.

The salnpling step retrieves a set of webpages that contain the query terrns, using sorne traditional technique. For exarnple, 'we can evaluate the query as a boolean keyword search and retrieve all webpages that contain the query terrns. We call the resulting set of pages the root set. The root set Inight not contain all relevant pages because senne authoritative pages rnight not include the user query \vords. But \ve expect that at least SOlne of the pages in the root set contain hyperlinks to the rnost relevant authoritative pages or that SCHne authoritative pages link to pages in the root set. This rnotivates our notion of a link page. We call a page a link page if it has a hyperlink to sorne page in the root set or if a page in the root set has a hyperlink to it. In order not to Iniss potentially relevant pages, we auglnent the root set by all link pages and we call the resulting set of pages the base set. Thus, the base set includes all root pages and all link pages; we refer to a webpage in the base set as a base page.

Our goal in the second step of the algorithrn is to find out which base pages are good hubs and good authorities and to return the best authorities and hubs as the answers to the query. To quantify the quality of a base page as a hub and as an authority, we associate with each base page in the base set a hub weight and an authority weight. The hub weight of the page indicates the quality of the page as a hub, and the authority weight of the page indicates the quality of the page as an authority. We cornpute the weights of each page according to the intuition that a page is a good authority if rnany good hubs have hyperlinks to it, and that a page is a good hub if it has rnany outgoing hyperlinks to good authorities. Since we do not have any a priori knowledge about which pages are good hubs and authorities, we initialize all weights to one. We then update the authority and hub weights of base pages iteratively as described below.

Consider a base page p with hub weight h_p and with authority weight $a_{p'}$ In one iteration, we update a_p to be the SUlU of the hub weights of all pages that have a hyperlink to p. Formally:

$$a_p = \sum_{\text{All base pages } q \text{ that have a link to } p} h_q$$

Analogously, we update h_p to be the SUlll of the weights of all pages that p points to:

$$h_p = \sum_{\text{All base pages } q \text{ such that } p \text{ has a link to } q} a_q$$

Cornparing the algorithrn with the other approaches to querying text that we discussed in this chapter, we note that the iteration step of the HITS algorithm—the distribution of the weights·· does not take into a,ccount the words on the base pages. In the iteration step, we are only concerned about the relationship between the base pages as represented by hyperlinks.

The lIlTS algorithrIl usually produces very good results. For exarnple, the five highest ranked results frorn Google ("rhich uses a variant of the HITS algorithrn) far the query 'Raghu Ramakrishnan' are the following webpages:

```
www.cs.wisc.edu/~raghu/raghu.html
www.cs.wisc.edu/~dbbook/dbbook.html
www.informatik.uni-trier.de/
    ~ley/db/indices/a-tree/r/Ramakrishnan:Raghu.html
www.informatik.uni-trier.de/
```

Computing bub and authority weights: We can use matrix notation to write the updates for all hub and authority weights in one step. Assume that we nUluber aU pages in the base set $\{1, 2, ..., n\}$. The adjacency matrix B of the base set is an n x n matrix whose entries are either Oor 1. The rnatrix entry (i, j) is set to 1 if page i has a hyperlink to page j; it is set to 0 otherwise. We can also write the hub weights h and authority weights a in vector notation: $h = \langle h_1, \ldots, h_n\rangle$ and $a = \langle$al,'" $, a_n\rangle$. We can now rewrite our upda,te rules as follo\vs:

$$h = B \cdot a, \quad \text{and} \quad a = B^T \cdot h .$$

Unfolding this equation once, corresponding to the first iteration, we obtain:

$$h = BB^T h = (BBT)h, \quad \text{and} \quad a = BTBa = (BTB)a .$$

After the second iteration, we arrive at:

$$h = (BB^T)^2 h, \quad \text{and} \quad a = (B^T B)^2 a .$$

Results from linear algebra tell us that the sequence of iterations for the hub (resp. authority) weights converges to the principal eigenvectors of BET (resp. $B^T B$) if we normalize the weights before each iteration so that the suru of the squares of all weights is always $2 \cdot n$. Furthermore, results from linear algebra tell us that this convergence is independent of the choice of initial weights, as long as the initial weights are positive. Thus, our rather arbitrary choice of initial weights··--we initialized all hub and authority weights to 1—does not change the outcolne of the algorithm.

Google's Pigeon Rank: Google corIlputes the *pigeon rank (PRJ* for a webpage A using the following forrIlula, which is very sirnilar to the H.ub-Authority ranking functions:

$$PR(A) = (1 - d) + d(PR(T_1)/C(T_1) + \ldots + PR(T_n)/C(T_n))$$

'$T_1 \ldots Tn$ are the pages that link (or 'point') to A, $C(T_i)$ is the rllllnber of links going out of page T_i, and d is a heuristically chosen constant (Google uses 0.85). Pigeon ranks fofm a probability distribution over all webpages; the Slun of ranks over all pages is 1. If we consider a rnodel of user behavior in which a user randornly chooses a page and then repeatedly clicks on links until he gets bored and randoll1ly chooses a new page, the probability that the user visits a page is its Pigeon rank. The pages in the result of a search are ranked using a cornbination of an IR-style relevance l11etric and Pigeon rank.

SQL/MM: Full Text 'Full text' is described as data that can be searched, unlike simple character strings, and a new data type called FullText is introduced to support it. The Inethods associated 'with this type support searching for individual words, phrases, words that 'sound like' a query terlll, etc. Three 11lethods are of particular interest. CONTAINS checks if a FullText object contains a specified search terln (word or phrase). RANK returns the relevance rank of a FullText object with respect to a specified search terln. (I-Iow the rank is defined is left to the hnplementation.) IS ABOUT detennines whether the FullText object is sufficiently related to the specified search term. (The behavior of IS ABOUT is also left to the iInplelllentation.)

Relational DBMSs from IBIvI, Microsoft, and Oracle all support text fields, although they do not currently conforrll to the SQL/MM standard.

-ley/db/indices/a-tree/s/Seshadri:Praveen.html
www.acm.org/awards/fellows_citations_n-z/ramakrishnan.htmI

The first result is Rarnakrishnan's horne page; the second is the horne page for this book; the third is the page listing his publications in the popular DBLP bibliography; and the fourth (initially puzzling) result is the list of publications for a forrner student of his.

27.5 MANAGING TEXT IN A DBMS

In preceding sections, we saw how large text collections are indexed and queried in JR, systerns and Web search engines. We now consider the additional challenges raised by integrating text data into database systerns.

The basic approach being pursued by the SQI.I standards cornrnunity is to treat text docllrnents as a new data type, FullText, that can appear as the value of a field in a table. If we define a table with a single cohunn of type FullText, each row in the table corresponds to a docurnent in a dOClllnent collection. IVlethods of FullText can be llsed in the WHERE clause of SQL queries to retrieve rows containing text objects that Inatch an IR-style search criterion. The relevance rank of a FullText object can be explicitly retrieved using the RANK rnethod, and this can be llsed to sort results by relevance.

Several points ruust be kept in rnind as we consider this approach:

- This is an extremely general approach, a,nel the perforlnance of a SQL system that supports such an extension is likely to be inferior to a specialized IR SystCIIl.

- The rnodel of data does not ad.equately reflect docurnents with additional rnetadata. If we store docurnents in a table with a FullText colurnn and use additional cohl1nns to store rnetadata--for exarnple, author, title, SUIll-Inary, rating, popularity—relevance rneasures that cornbine nletadata 'with IR similarity measures 11lUSt be expressed using lle\V user-defined rneth-ods, because the RANK rnethod only has access to the FullText object, and not the rnetadata. The ernergence of XML docurnents, which have non-uniforrn, partial rlletadata, further cornplicates nlatters.
- The handling of updates is unclear. As we have seen, IR indexes are corll-plex, and expensive to 111aintain. Requiring a systern to update the indexes before the updating transaction cOl1ullits can irnpose a severe perfonnance penalty.

27.5.1 Loosely Coupled Inverted Index

The irrlplenlcntation approach used in current relational DBMSs that support text fields is to have a separate text-search engine that is loosely coupled to the DBMS. The engine periodically updates the indexes, but provides no transac-tional guarantees. Thus, a transaction could insert (a row containing) a text object and cornrnit, and a subsequent transaction that issues a Inatching search might not retrieve the (row containing the) object.

27.6 A DATA MODEL FOR XML

.Aswe saw in Section 7.4.1, XML provides a way to rnark up a docurnent with rneaningful tags that irnpart SaIne partial structure to the docurnent. *Semistructured data rnodels,* which we introduce in this section, capture rnuch of the structure in XML doculnents, while abstracting away Inany deta.ils.[1] Sernistructured data Inodels have the potential to serve as a forInal foundation for XlVIL and enable us to rigorously define the sernantics of queries over XlVIL, which we discuss in Section 27.7.

27.6.1 Motivation for Loose Structure

Consider a set of doculnents on the Web that contain hyperlinks to other doc-UHlents. These docurnents, although not eornpletely unstructured, cannot be rnodeled naturally in the relational data rnodel because the pattern of hyper-links is not regular across docurnents. In fact, every HTML docurnent has

1.An iruportant aspect of XML tha.t is *not* captured is the ordering of elements. A more complete data model called XData has been proposed by the W3C committee that is developing XML standards, but we do not discuss it here.

XML Data Models: 'A number of data lnodels for XML are being considered by standards COilllnittees such as ISO and W3C. W3C's Infoset is a tree-structured model, and each node can be retrieved through an accessor function. A version called Post-Validation Infoset (PSVI) serves as the data model for XML Schelna. TheXQuery language has yet another data model associated with it. The plethora of l110dels is due to parallel developrnent in SOllle cases, and due to different objectives in others. Nonetheless, all these nlodels have loosely-structured trees as their central feature.

some minirnal structure, such as the text in the TITLE tag versus the text in the doculnent body, or text that is highlighted versus text that is not. As another example, a bibliography file also has a certain degree of structure due to fields such as *author* and *title,* but is otherwise unstructured text. Even data that is 'unstructured', such as free text or an ilnage or a video clip, typically has some associated information such as timestamp or author infornlation that contributes partial structure.

We refer to data with such partial structure as semistructured data. There are rnany reasons why data might be semistructured. First, the structure of data nlight be irnplicit, hidden, unknown, or the user Inight choose to ignore it. Second, when integrating data froln several heterogeneous sources, data exchange and transforrnation are inlportant problerns. We need a highly flexible data rnodel to integrate data froIn all types of data sources including flat files and legacy systenls; a structured data model such as the relational rnodel is often too rigid. Third, we cannot query a structured database without knowing the scheIna, but sOlnetimes we want to query the data without full knowledge of the scherna. For exarnple, we cannot express the query "Where in the database can we find the string *Malgudi*?" in a relational database systern \vithout knowing the schcrna, and knowing which fields contain such text values.

27.6.2 A Graph Model

All data rnodels proposed for sernistrnctured data represent the data as scnne kind of labeled graph. Nodes in the graph correspond to cornpound objects or atornic values.. Each edge indicates an object-subobject or object-value relationship. Leaf nodes, i.e, nodes with no outgoing edges have a value a.ssociatecl \vith thern. rrhere is no separate scherna and no auxiliary description; the data in the graph is self describing. For exarnple, consider the graph shown in Figure 27.7, which represents part of the XML data f1'01n Figure 7.2. The root node of the graph represents the outennost elernent, BOOKLIST. The node has three children that are labeled with the elClnent narne BOOK, since the list of books

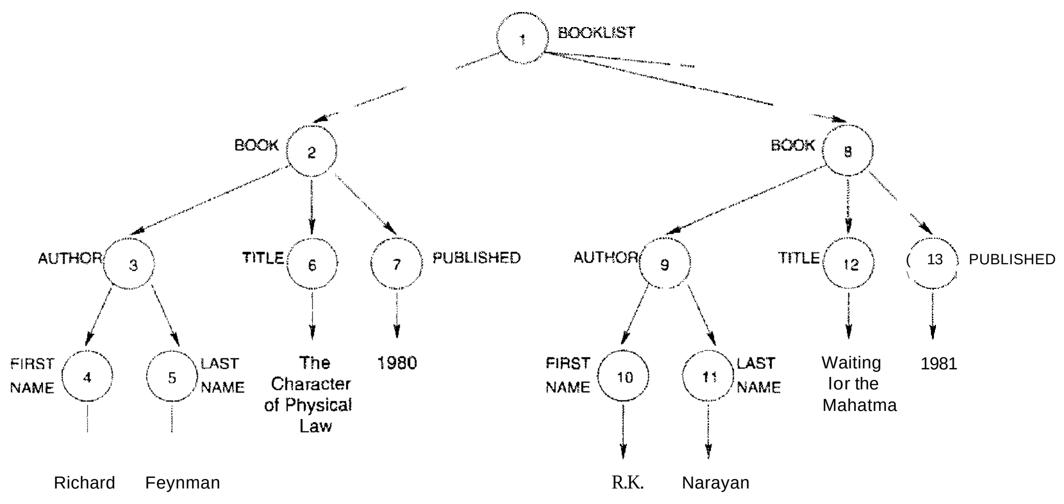

Figure 27.7 The Semistructured Data Model

consists of three individual books. The numbers within the nodes indicate the object identifier associated with the corresponding object.

We now describe one of the proposed data models for semistructured data, called the **object exchange model (OEM)**. Each object is described by a quadruple consisting of a *label,* a *type,* the *value* of the object, and all. *object identifier* which is a unique identifier for the object. Since each object has a label that can be thought of as a column nallle in the relational model, and each object has a type that can be thought of as the column type in the relational rnodel, the object exchange Illodel is self-describing. Labels in OEM should be as infol'rnative as possible, since they serve two purposes they can be used to identify an object as well as to convey the meaning of an object. For example, we can represent the last Halne of an author as follows:

(lastName, string, "Feynman")

More cOInplex objects are decornposed hierarchically into srnaller objects. For exalllple, a,n author nalne can contain a first narne and a last narne. rrhis object is described as follows:

(authorName, set, $\{fiTstname_1, lastnaTnel\}$)
  $firstname_1$ is (firstName, string, "Richard")
  $lastname_{1'}$ is (lastName, string, "Feynman")

As another exarnple, an object representing a set of books is described as follows:

(bookList, set, $\{book_1, book_2, book_3\}\rangle$
  book} is (book, set, $\{author_1, title\}, published_1\}\rangle$

SQL and XML: XQuery is a standard proposed by the World-Wide Web Consortiurn (W3C). In parallel, standards con:unittees developing the SQL standards have been working on a successor to SQL:1999 that supports XML. The part that relates to XML is tentatively called SQL/XML and details can be found at http: / / sqlx . arg.

$book_2$ is ⟨book, set, $\{author_2, title_2, published_2\}$⟩
book3 is ⟨book, set, {$author_3$, *t'itle3,Published3})*
  authoT3 is (author, set, *{!'lrstnarnJe3,* $lastname_3$})
  t'itle3 is (title, string, liThe English Teacher")
  $published_3$ is (published, integer, 1980)

27.7 XQU'ERY: **QUERYING XML DATA**

Given that XlvII.. doculnents are encoded in a way that reflects (a considerable amount of) structure, we have the opportunity to use a high-level language that exploits this structure to conveniently retrieve data fro111 within such documents. Such a language would also allow us to easily translate XML data between different DTDs, as we lllUSt when integrating data from multiple sources. At the tirne of writing of this book, XQuery is the W3C standard query language for XML data. In this section, we give a brief overview of XQuery.

27.7.1 **Path Expressions**

Consider the XlvII.. dOCUlnent shown in Figure 7.2. The following exarnple query returns the last nanles of all authors, assullling that our XML docurnent resides at the location www.ourbookstore.com/books .xml.

```
FOR
   $1 IN doc(www.ourbookstore.com/books.xml)//AUTHOR/LASTNAME
RETURN <RESULT> $1 </RESULT>
```

This exarnple illustrates sonle of the basic constructs of XQuery. The FOR clause in XQuery is roughly analogous to the FROM clause in SQL. The RETURN clause is sirnilar to the SELECT clause. We return to the general fornl of queries shortly, after introducing an irnportant concept called a path expression.

The expression

```
doc(www.ourbookstore.com/books.xml)//AUTHOR/LASTNAME
```

XPath and Other XML Query Languages: Path expressions in XQuery are derived frorn XPath, an earlier XML query facility. Path expressions in XPath can be qualified with selection conditions, and can utilize several built-in functions (e.g., counting the nurnber of nodes rnatched by the expression). Many of XQuery's features are borrowed {roIn earlier languages, including XML-QL and Quilt.

in the FOR clause is an exarnple of a path expression. It specifies a path involving three entities: the docurnent itself, the AUTHOR elernents and the LASTNAME elernents.

The path relationship is expressed through separators / and //. The separator // specifies that the AUTHOR elernent can be nested anywhere within the document whereas the separator / constrains the LASTNAME elernent to be nested immediately under (in terms of the graph structure of the docurnent) the AUTHOR element. Evaluating a path expression returns a *set* of elernents that rnatch the expression. The variable *l* in the example query is bound in turn to each LASTNAME elernent returned by evaluating the path expression. (To distinguish variable naInes fronl normal text, variable narnes in XQuery are prefixed with a dollar sign $.)

The RETURN clause constructs the query result-·--which is also an XML docurnent-···_· by bracketing each value to which the variable *l* is bound with the tag RESULT. If the exanlple query is applied to the sarnple data shown in Figure 7.2, the result would be the following XML docurnent:

```
<RESULT><LASTNAME>Feynman </LASTNAME></RESULT>
<RESULT><LASTNAME>Narayan </LASTNAME></RESULT>
```

We use the docurnent in Figure 7.2 as our input in the rest of this chapter.

27.7.2 FLWR Expressions

The basic fornl of an XQuery consists of a FLWR **expression**, where the letters denote the FOR, LET, WHERE and RETURN clauses. The FOR and LET clauses bind variables to values through path expressions. These values are qualified by the WHERE clause, and the result XML fragrnent is constructed by the RETURN clause.

rrhe difference between a FOR and LET clause is that while FOR binds a variable to each elernent specified by the path expression, LET binds a variable to the whole *collection* of elernents. Thus, if we change our exarnple query to:

```
LET
   $IINdoc(www.ourbookstore.com/books.xm1)//AUTHOR/LASTNAME
RETURN <RESULT> $1 </RESULT>
```

then the result of the query beconles:

```
<RESULT>
   <LASTNAME>Feynman</LASTNAME>
   <LASTNAME>Narayan</LASTNAME>
</RESULT>
```

Selection conditions are expressed using the WHERE clause. Also, the output of a query is not lirnited to a single elernent. These points are illustrated by the following query, which finds the first and last names of all authors who wrote a book that was published in 1980:

```
FOR $b IN doc(www.ourbookstore.com/books.xm1)/BOOKLIST/BOOK
WHERE $b/PUBLISHED='19S0'
RETURN
   <RESULT> $b/AUTHOR/FIRSTNAME, $b/AUTHOR/LASTNAME </RESULT>
```

The result of the above query is the following XML docurnent:

```
<RESULT>
   <FIRSTNAME>Richard </FIRSTNAME><LASTNAME>Feynman </LASTNAME>
</RESULT>
<RESULT>
   <FIRSTNAME>R.K. </FIRSTNAME><LASTNAME>Narayan </LASTNAME>
</RESULT>
```

For the specific DTI) in this exalnple, where a BOOK elernent has only one AUTHOR, the above query can be written by using a different path expression in the FOR clause, as follows.

```
FOR $a IN
   doc(www.ourbookstore.com/books.xml)
      /BOOKLIST/BOOK[PUBLISHED='19S0']/AUTHOR
RETURN <RESULT> $a/FIRSTNAME, $a/LASTNAME </RESULT>
```

rrhe path expression in this query is an instance of a branching path expression. The variable / is now bound to every AUTHOR elernent that rnatches the path doc/BOOKLIST/BOOK/AUTHOR where the intennediate BOOK elClnent is constrained to have a PUBLISHED elernent nested inunediately within it with the value 1980.

27.7.3 Ordering of Elements

XML data consists of *ordered* doculnents and so the query language IllUSt return data in SOUle order. The selnantics of XQuery is that a path expression returns results sorted in document order. Thus, variables in the FOR clause are bound in doculnent order. If however, we desire a different order, we can explicitly order the output as shown in the follo\ving query, which returns TITLE elernents sorted lexicographically.

```
FOR
   $b IN doc(www.ourbookstore.com/books.xml)/BOOKLIST/BOOK
RETURN <BOOKTITLES> $b/TITLE </BOOKTITLES>
SORT BY TITLE
```

27.7.4 Grouping and Generation of Collection Values

Our next example illustrates grouping in XQuery, which allows us to generate a new collection value for each group. (Contrast this with grouping in SQL, which only allows us to generate an aggregate value (e.g., SUM) per group.) Suppose that for each year we want to find the last narnes of authors who wrote a book published in that year. We group by year of publication and generate a list of last names for each year:

```
FOR $p IN DISTINCT
   doc(www.ourbookstore.com/books.xml)/BOOKLIST/BOOK/PUBLISHED
   RETURN
   <RESULT>
      $p,
      FOR $a IN DISTINCT /BOOKLIST/BOOK[PUBLISHED=$pJ/AUTHOR
         RETURN $a
   </RESULT>
```

The keyword *DISTINCT* elirninates duplicates fronl the collection returned by a, path expression. Using the XML docurnent in Figure 7.2 as input, the above query produces the following result:

```
<RESULT> <PUBLISHED>1980</PUBLISHED>
   <LASTNAME>Feynman</LASTNAME>
   <LASTNAME>Narayan</LASTNAME>
</RESULT>
<RESULT> <PUBLISHED>1981</PUBLISHED>
   <LASTNAME>Narayan</LASTNAME>
</RESULT>
```

27.8 EFFICIENT EVALUATION OF XML **QUERIES**

X.Query operates on XML data and produces XTvIL data as output. In order to be able to evaluate queries efficiently, we need to address the follo\ving issues.

- Storage: We can use an existing storage systerIl like a relational or object oriented systerll or design a new storage forInat for XML doclunents. There are several ways to use a relational systenl to store XML. One of thern is to store the XML data as Character Large Objects (CLOBs). (CLOBS were discussed in Chapter 23.) In this case, however, we cannot exploit the query processing infrastructure provided by the relational systerrl and would instead have to process XQuery outside the database systenl. In order to circumvent this problenl, we need to identify a scherna according to which the XML data can be stored. These points are discussed in Section 27.8.1.

- Indexing: Path expressions add a lot of richness to XQuery and yield lllany new access patterns over the data. If we use a relational system for storing XML data, then we are constrained to use only relational indexes like the B-Tree. However, if we use a native storage engine, then we have the option of building novel index structures for path expressions, some of which are discussed in Section 27.8.2.

- Query Optimization: Optirnization of queries in XQuery is an open problern. The work so far in this area can be divided into three parts. 'rhe first is developing an algebra for XQuery, analogous to relational algebra. The second research direction is providing statistics for path expression queries. Finally, SOlne work has addressed sirnplification of queries by exploiting constraints on the data. Since query optirnization for X.Query is still at a prelirninary stage, we do not cover it in this chapter.

Another issue to be considered while designing a new storage systeul for XML data is the verbosity of repeated tags. As we see in Section 27.8.1) using a relational storage systelu addresses this problern since tag narnes are not stored repeatedly. If on the other hand, we want to build a native storage systcrn, then the rnanner in which the XML data is cornpressed becornes significant. Several cornpression. algorithrHs are known that achieve cOlnpression ratios close to relational storage, 1n1t we do not discuss ther11 here.

27.8.1 Storing XML in RDBMS

One natural candidate for storing XML data is a relational database systern. The ruain issues involved in storing XML data in a relational systelTI are:

Commercial database systems and XML: Many relational and object-relational database systerll vendors are currently looking into support for XML in their database engines. Several vendors of object-oriented database Inanagenlent systems already offer database engines that can store XML data whose contents can be accessed through graphical 11ser interfaces orJl server-side Java extensions.

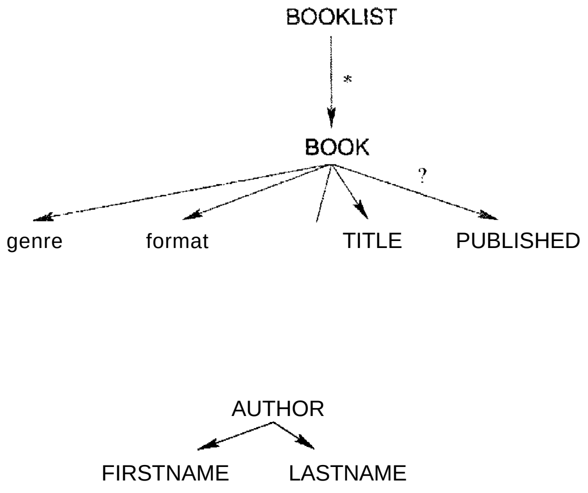

Figure 27.8 Bookstore XML DTD Element Relationships

- *C;hoice of relational schema*: In order to use an RDBMS, we need a scherna. What relational schema should we use even assuming that the XML data COUles with an associated scherna?
- *Queries:* Queries on XML data are in XQuery whereas a relational systern can only handle SQL. Queries in XQuery therefore need to be *translated* into SQL.
- *Reconstruction*: rrhe output of XQuery is XML. Thus, the result of a SQL query needs to be converted back into XML.

Mapping XML Data to Relations

We illustrate the rnapping process through our bookstore exarnple. rrhe nesting rela,tionships among the different elernents in the DTD is shown in Figure 27.8. The edges indicate the nature of the nesting.

()ne way to derive a l'clation.al schelna is as follows. We begin at the BOOKLIST elernent and create a relation to store it. rrraversing down froIn BOOKLIST, we get BOOK following a * edge. This edge indicates that we store the BOOK elernents in a separate relation. Traversing further down, we see that all elcrnents and attributes nested within BOOK occur at 1I10St once. Hence, we can store thenl in the saIne relation as BOOK. The resulting relational schelIla *Relschernal* is shown below.

BOOKLIST(*id:* integer)
BOOK (*booklistid:* integer, *author_firstname:* string,
  author_lastnarne: string, *title:* string,
  published: string, *genre:* string, *format:* string)

BOOK. *booklistid* connects BOOK to BOoKLIST. Since a DTD has only one base type, string, the only base type used in the above schelna is string. The constraints expressed through the DTD are expressed in the relational schema. For instance, since every BOOK must have a TITLE child, we Illust constrain the *title* column to be *non-null.*

Alternatively, if the DrrD is changed to allow BOOK to have more than one AUTHOR child, then the AUTHOR elements cannot be stored in the sallie relation as BOOK. This change yields the following relational schema *Relschema2.*

BOOKLIST(*id:* integer)
BOOK (*id:* integer, *booklistid:* integer,
  title: string, *published:* string, *genre:* string, *for1nat:* string)
AUTHoR(*bookid:* integer, *firstname:* string, *lastname:* string)

The column AUTHOR. *bookid* connects AUTHOR to BOOK.

Query Processing

Consider the following example query again:

```
FOR
   $b IN doc(www.ourbookstore.com/books.xml)/BOOKLIST/BooK
WHERE $b/PUBLISHED='1980'
RETURN
   <RESULT> $b/AUTHOR/FIRSTNAME, $b/AUTHOR/LASTNAME </RESULT>
```

If the nlapping between the XML data and relational tables is known, then we can construct a SQL query that returns all colunlIls that are needed to reconstruct the result XML docuIllent for this query. Conditions enforced by the path expressions and the WHERE clause are translated into equivalent conditions in the SQL query. We obtain the following equivalent SQL query if we use *Relschema1* as our relational scherna.

```
SELECT BOOK. author_firstname, BOOK. author_lastname
```

```
FROM    BOOK, BOOKLIST
WHERE   BOOKLIST.id = BOOK.booklistid
           AND BOOK.published='1980'
```

The results thus returned by the relational query processor are then tagged, outside the relational systern, as specified by the RETURN clause. This is the result of the *reconstruct'ion* phase.

In order to understand this better, consider what happens if we allow a BOOK to have 111ultiple AUTHOR children. Assume that we use *Rel8chema2* as our relational schema. Processing the FOR and WHERE clauses tells us that it is necessary to join relations BOOKLIST and BOOK with a selection on the BOOK relation corresponding to the year condition in the above query. Since the RETURN clause needs information about AUTHOR elements, we need to further join the BOOK relation with the AUTHOR relation and project the *jir8tname* and *lastname* columns in the latter. Finally, since each binding of the variable $b in the above query produces one RESULT element, and since each BOOK is now allowed to have more than one AUTHOR, we need to project the *id* column of the BOOK relation. Based on these observations, we obtain the following equivalent SQL query:

```
SELECT    BOOK.id, AUTHOR. firstname , AUTHOR.lastname
FROM      BOOK, BOOKLIST, AUTHOR
WHERE     BOOKLIST.id = BOOK.booklistid AND
             BOOK.id = AUTHOR.bookid AND BOOK.published='1980'
GROUP BY  BOOK.id
```

T'he result is grouped by BOOK.id. The tagger outside the database system now receives results clustered by the BOOK element and can tag the resulting tuples on the fly.

Publishing Relational Data as XML

Since XML has elnerged as the standard data exchange fornat for business applications, it is necessary to publish existing business data as XML. Most operational business data is stored in relational systerns. Consequently, 111ech-anisrns have been proposed to publish such data as XML docul11ents. These involve a language for specifying henv to tag and structure relational data and an irnplernentation to carry out the conversion. This 111apping is in some sense the reverse of the XML-to-relationaJ rnapping used to store XML data. 'The conversion process Inirnics the reconstruction phase when we execute XQuery using a relational systern. The published XML data can be thought of as an XML view of relational data. This view can be queried using XQuery. One lnethod of executing XQuery on such vie'ws is to translate thCIIl into SQL and thCIl construct the XML result.

27.8.2 Indexing XML Repositories

Path expressions are at the heart of all proposed XIVIL query languages, in particular XQuery. A natural question that arises is how to index XML data to support path expression evaluation. The ainl of this section is to give a flavor of the indexing techniques proposed for this probleul. We consider the OEM rnodel of senlistructured data, 'where the data is self-describing and there is no separate scherna.

Using a B+ Tree to Index Values

Consider the following XQuery exaluple, which we discussed earlier on the bookstore XML data in Figure 7.2. The OEM representation of this data is shown in Figure 27.7.

```
FOR
   $b IN doc(www.ourbookstore.com/books.xml)/BOOKLIST/BOOK
WHERE  $b/PUBLISHED='1980'
RETURN
   <RESULT> $b/AUTHOR/FIRSTNAME, $b/AUTHOR/LASTNAME </RESULT>
```

This query specifies joins alIlong the objects with labels BOOKLIST, BOOK, AUTHOR, FIRSTNAME, LASTNAME and PUBLISHED with a selection condition on PUBLISHED objects.

Let us suppose that we are evaluating this query in the absence of any indexes for path expressions. However, we do have a value index such as a B-T'ree that enables us to find the ids of all objects with label PUBLISHED and value 1980. There are several ways of executing this query under these assumptions.

For instance, we could begin at the docurncnt root and traverse down the data graph through the BOOKLIST object to the BOOK objects. By further traversing the data graph downwards, for each BOOK object we can check whether it satisfies the valuc'predicate (PUBLISHED='1980'). Finally, for those BOOK objects that satisfy the predicate, we can find the relevant FIRSTNAME and LASTNAME objects. This approach corresponds to a top-down evaluation of the query.

Alternatively, we could begin by using the value index to find all PUBLISHED ol)jects that satisfy PUBLISHED='1980'. If the data graph can be traversed in the reverse directiono·that is, given an object, we can find its parent—then we

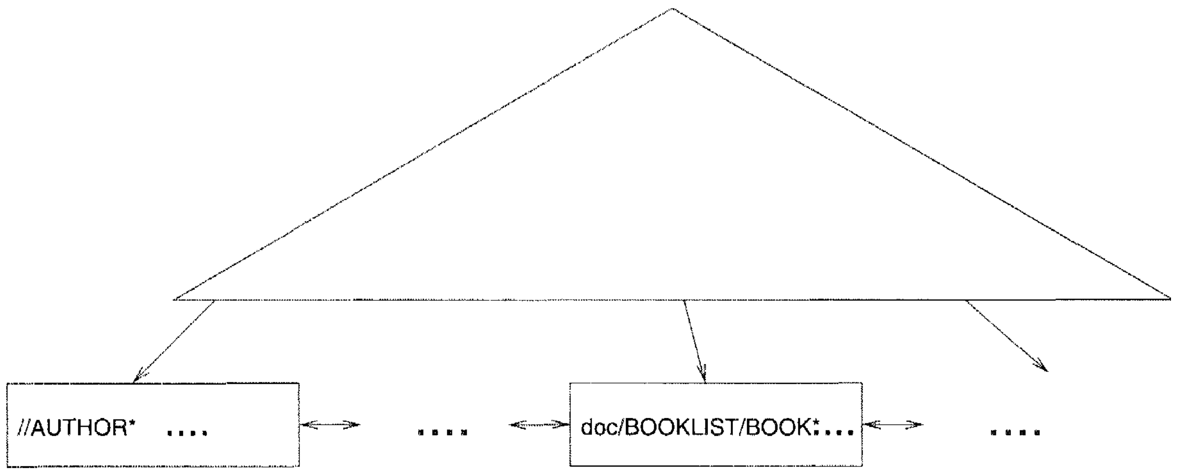

Figure 27.9 Path Expressions in a B-Tree

can find all parents of the PUBLISHED objects retaining only those that have label BOOK. We can continue in this manner until we find the FIRSTNAME and LASTNAME objects of interest. Observe that we need to perforrll all joins in the query on the fly.

Indexing on Structure vs. Value

Now let us ask ourselves whether traditional indexing solutions like the B-Tree can be used to index path expressions. We can use the B-Tree to rllap a path expression to the ids of all objects returned by it. The idea is to treat all path expressions as strings and order therIl lexicographically. Every leaf entry in the B-Tree contains a string representing a, path expression and a list of ids corresponding to its result. Figure 27.9 shows how such a B-Tree would look. Let us contrast this with the traditional problern of indexing a well-ordered dornain like integers for point queries. In the latter case, the nurnber of distinct point queries that can be posed is just the rnllnber of data values and so is linear in the data size.

The scenario with path indexing is fundarnentally different—the variety of ways in which we can cornbine tags to forrn (sirnple) path expressions co11-pled with the power of placing // separators leads to a rnuch larger nurnber of possible path expressions. For instance, an AUTHOR clcrnent in the exarn-pIe in Figure 27.7 is returned as part of the qllcries BOOKLIST/BOOK/AUTHOR, //AUTHOR, //BOOK//AUTHOR, BOOKLIST//AUTHOR and so o11. The nurnber of distinct queries can in fact be exponential in the data size (lneasured in tenns of the rnunber of XIVIL elelnents) in the worst case. This is \vhat rnotivates the search for alternative strategies to index path expressions.

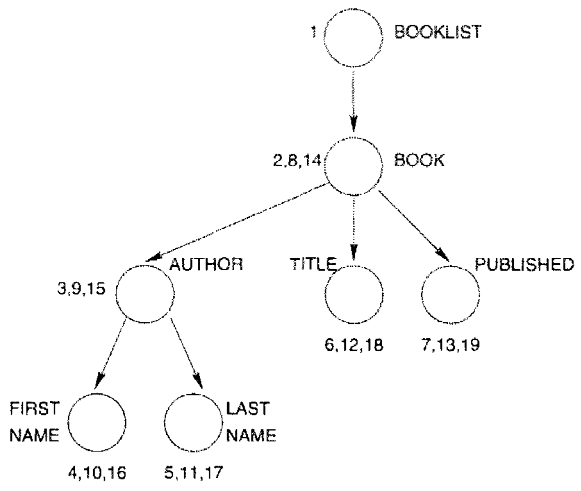

Figure 27.10 Example Path Index

The approach taken is to represent the mapping between a path expression and its result by means of a structural sunlIllary which takes the fornl of another labeled, directed graph. rrhe idea is to preserve all the paths in the data graph in the surllrnary graph, while having far fewer nodes and edges. An extent is associated with each node in the SUllllnary. The extent of an index node is a subset of the data nodes. The surnmary graph along with the extents constitutes a path index. A path expression is evaluated using the index by evaluating it against the sumrnary graph and then taking the union of the extents of all rnatching nodes. This yields the index result of the path expression query. The index covers a path expression if the index result is the eorrect result; obviously, we can use an index to evaluate a path expression only if the index covers it.

Consider the structural SUlnrnary shown in Figure 27.10. rrhis is a path index for the data in Figure 27.7. Tlhe nurnbers shown beside the nodes correspond to the respective extents. Let us now exarnine how this index can change the top-down evaluation of the exaruple query used earlier to illustrate B+ tree value indexes.

rrhe top-down evaluation as outlined above begins at the docurnent root and traverses down to the BOOK objects. rrhis can be achieved rnore efficiently by the path index. Instead of traversing the data graph, we can traverse the path index down to the BOOK object in the index and look up its extent, which gives us the ids of all BOOK objects that rnatch the path expression in the FOR clause. The rest of the evaluation then proceeds as before. Thus, the path index saves us frorn perfonning joins by essentially precorIlputing thern. We note here that the path index shown in Figure 27.10 is isornorphic to the DTD schcIlla graph ShO\vII in Figure 27.8. This drives horne the point that the path index \vithout the extents is a structural SUHllnary of the data.

rrhe ahove path index is the Strong Dataguide. If we treat path expressions as strings, then the dataguide is the trie representing thern. The trie is a well-known data structure used to search regular expressions over text. This shows the deeper unity between the research on indexing text and the XML path indexing work. Several other path indexes have been also proposed for senli-structured data, and this is an active area of research.

27.9 REVIEW QUESTIONS

Answers to the review questions can be found in the listed sections.

- What is information retrieval? (Section 27.1)
- What are some of the differences between DBMS and IR systems? Describe the differences between a ranked query and a boolean query. (Section 27.2)
- What is the vector space model, and what are its advantages? (Section 27.2.1)
- What is TF/IDF terrn weighting, and why do we weigh by both? We do we eliminate stop words? What is length norrnalization, and why is it done? (Section 27.2.2)
- How can we measure document similarity? (Sections 27.2.3)
- What are *precision* and *recall,* and how do they relate to each other? (Section 27.2.4)
- Describe the following two index structures for text: Inverted index and signature file. What is a bit-sliced signature file? (Section 27.3)
- How are web search engines architected? Ilow does the "hubs and authorities" a.lgorithrn work? Can you illustrate it on a srnall set of pages? (Section 27.4)
- What support is there for rnanaging text in a DBMS? (Section 27.5)
- Descibe the OEM data rnodel for sernistructured data. (Section 27.6)
- What are the elernents of XQuery? What is a path expression? What is an FLWR expression? How can we order the output of query? flow do we group query outputs? (Section 27.7)
- Describe how XTvlL data can be stored in a relational DBMS. How do we ma,p XML data to relations? Can we use the query processing infrastructure of the relational DBIvIS? How do 'we publish relational data as XML? (Section 27.8.1)

- I-Iow do we index collections of XML doeunlents? What is the difference between indexing on structure versus indexing on value? What is a path index'? (Section 27.8.2)

EXERCISES

Exercise 27.1 Carry out the following tasks.

1. Given an ASCII file, cOInpute the frequency of each word and create a plot siInilar to Figure 27.3. (Feel free to use public dornain plotting software.) Run the prograIll on the collection of files currently in your directory and see whether the distribution of frequencies is Zipfian. How can you use such plots to create lists of stop words?
2. The Porter stenliller is widely used, and code irnplernenting it is freely available. Download a copy, and run it on your collection of docuInents.
3. One criticisIn of the vector space nlodel and its use in sirnilarity checking is that it treats tenns as occurring independently of each other. In practice, Inany words tend to occur together (e.g., ambulance and emergency). Write a program that scans an ASCII file and lists all pairs of words that occur within 5 words of each other. For each pair of words, you now have a frequency, and should be able to create a plot like Figure 27.3 with pairs of words on the X-axis. Run this program on some sample docuIll€nt collections. What do the results suggest about co-occurrences of words?

Exercise 27.2 Assunle you are given a docurnent database that contains SIX documents. After stemming, the docurnents contain the following ternlS:

Document	Terrns
1	car rnanufacturer Honda auto
2	auto cornputer navigation
3	Honda navigation
4	11lanufactllrer cOlnputer IBM
5	IBNI personal cOInputer
6	car Beetle VW

Answer the following questions.

1. 8ho\v the result of creating an inverted file on the docurncnts.
2. Show the result of creating a signature file with a width of 5 bits. Construct your own hashing function that rnaps terms to bit positions.
3. Evaluate the following boolea.n queries using the inverted file and the signature file that you created: 'car', 'IBM' AND 'COIuputer', 'IBM' AND 'car', 'IBM' OR 'auto', and 'IBM' AND 'computer' AND 'rnanufacturer'.
4. Assurne that the query loacl against the docurnent databa.se consists of exactly the queries that were stated in the previous question. Also assurne that each of these queries is evaluated exactly oncc.
   (a) Design a signature file with a width of 3 bits and design a hashing function that minimizes the overall nurnber of false positives retrieved when evaluating the

(b) Design a signature file with a width of 6 bits and a hashing function that minimizes the overall nUlnber of false positives.

(c) Assume you want to construct a signature file. What is the sInallest signature width that allows you to evaluate all queries without retrieving any false positives?

5. Consider the following ranked queries: 'car, 'IBM COIllputer', 'IBM car', 'IBM auto', and 'IBM COIllputer rnanufacturer'.

   (a) Calculate the IDF for every tenn in the database.

   (b) For each doculnent, show its doctunent vector.

   (c) For each query, calculate the relevance of each doclunent in the database, with and without the length norrnalization step.

   (d) Describe how you would use the inverted index to identify the top two documents that Illatch each query.

   (e) How would having the inverted lists sorted by relevance instead of document id affect your answer to the previous question?

   (f) Replace each docurnent with a variation that contains 10 copies of the same document. For each query, recompute the relevance of each document, with and without the length normalization step.

Exercise 27.3 Assume you are given the following steIlllned docurnent database:

Document	Terms
1	car car IIlanufacturer car car Honda auto
2	auto computer navigation
3	Honda navigation auto
4	manufacturer computer IBl\II graphics
5	IBM personal IBM computer IBl\II IBl\I! IBM IBM
6	car Beetle VW Honda

Using this database, repeat the previous exercise.

Exercise 27.4 You are in charge of the Genghis ('We execute fast') search engine. You are designing your server cluster to handle 500 Inillion hits a day and 10 billion pages of indexed data. Each rnachine costs $1000, and can store 10 million pages and respond to 200 queries per second (against these pages).

1. If you were given a budget of $500,000 dollars for purchasing Inachines, and were required to index all 10 billion pages, could you do it?

2. What is the IllinirIlurIl budget to index all pages? If you assurne that each query can be answered by looking at data in just one (10 rnillion page) partition, and that queries are uniformly distributed across partitions, what peak load (in nuruber of queries per second) can such a cluster handle?

3. How would your answer to the previous question change if each query, on average, accessed two partitions?

4. What is the ruinirlllnl1 budget required to handle the desired load of 500 rnillion hits per day if all queries are on a *single partition*? Assurne that queries are uniforrnly distributed with respect to tirTle of day.

5. How would your answer to the previous question change if the rllllnher of queries per day went up to 5 billion hits per day? How would it change if the number of pages went up to 100 billion'?
6. Assurne that each query accesses just one partition, that queries are ullifonnly distributed across partitions, but that at any given time, the peak load on a partition is upto 10 times the average load. What is the rniniIlHlnl budget for purchasing Inachines in this scenario?
7. Take the cost for rnachines [raIn the previous question and rnultiply it by 10 to reflect the costs of Illaintenance, adrninistration, network bandwidth, etc. This anlount is your annual cost of operation. Assume that you charge advertisers 2 cents per page. What fraction of your inventory (i.e., the total nUlllber of pages that you serve over the course of a year) do you have to sell in order to make a profit?

Exercise 27.5 Assume that the base set of the HITS algorithrn consists of the set of Web pages displayed in the following table. An entry should be interpreted as follows: Web page 1 has hyperlinks to pages 5 and 6.

Webpage	*Pages that this page has links to*
1	5, 6, 7
2	5, 7
3	6, 8
4	
5	1, 2
6	1, 3
7	1, 2
8	4

1. Run five iterations of the HITS algorithlll and find the highest ranked authority and the highest ranked hub.
2. Cornpute Google's Pigeon Rank for each page.

Exercise 27.6 Consider the following description of itelllS shown in the Eggface cornputer rnail-order catalog.

"Eggface sells hardware and software. We sell the new PalIn Pilot V for $400; its part nUlnber is 345. We also sell the IBM ThinkPad 570 for only $1999; its part nUIllber is 3784. We sell both business and entertainrnent software. I:vlicrosoft Office 2000 has just arrived and you can purchase the Standard Edition for only $140, part number 974; the Professional Edition is $200, part 975. '1'he new desktop publishing software from Adobe called InDesign is here for only $200, part 664. We carry the newest galnes from Blizzard software. You can start playing Diablo II for only $30, part nurnber 12, and yon can purchase Starcraft for only $10, part nlllIlber 812. Our goal is cornplete cllstorner satisfaction·····-·ifwe don't have what you want in stock, we'll give you SIO off your next purchase!"

1. Design an HTML docllrnent that depicts the itelIlS offered by Eggface.
2. Create a well-formed XML doculnent that describes the contents of the Eggface catalog.
3. Create a TYr.D for your XML docurnent and rnake sure that the docuJnent you created in the last question is valid with respect to this *1Y1'1)*,

4. Write an XQuery query that lists all software items in the catalog, sorted by price.
5. Write an XQuery query that, for each vendor, lists all software iterl1s froIn that vendor (i.e., one row in the result per vendor).
6. Write an XQuery query that lists the prices of all hardware items in the catalog.
7. Depict the catalog data in the semistructured data model as shown in Figure 27.7.
8. Build a dataguide for this data. Discuss how it can be used (or not) for each of the above queries.
9. Design a relational schellla to publish this data.

Exercise **27.7** A university database contains infonnation about professors and the courses they teach. The university has decided to publish this information on the Web and you are in charge of the execution. You are given the following information about the contents of the database:

In the fall sernester 1999, the course 'Introduction to Database Management Systems' was taught by Professor Ioannidis. The course took place Mondays and Wednesdays from 9–10 a.m. in room 101. The discussion section was held on Fridays fTOIn 9-10 a.m. Also in the fall semester 1999, the course 'Advanced Database Management Systems' was taught by Professor Carey. Thirty five students took that course which was held in room 110 Tuesdays and Thursdays from 1–2 p.m. In the spring semester 1999, the course 'Introduction to Database Management Systems' was taught by U.N. Owen on Tuesdays and Thursdays frOIn 3·_·-4 p.m. in room 110. Sixty three students were enrolled; the discussion section was on Thursdays from 4–5 p.m. The other course taught in the spring semester was 'Advanced Database Management Systems' by Professor Ioannidis, Monday, Wednesday, and Friday frorn 8-9 a.m.

1. Create a well-formed XML document that contains the university database.
2. Create a DTD for your XML docurnent. Make sure that the XML docurnent is valid with respect to this DTD.
3. Write an XQuery query that lists the names of all professors in the order they are listed on the Web.
4. Write an XQuery query that lists all courses taught in 1999. The result should be grouped by professor, with one row per professor, sorted by last narne. For a given professor, courses should be ordered by BaIIe and should not contain duplicates (Le., even if a professor teaches the sarne course twice in 1999, it should appear only once in the result).
5. Build a dataguide for this data. Discuss how it can be used (or not) for each of the above queries.
6. Design a relational schcrna to publish this data.
7. Describe the infonnation in a different XML document—a docurnent that has a different structure. Create a corresponding DTD and make sure that the docurnent is valid. Rc-fonnulate the queries you wrote for preceding parts of this exercise to work with the new DTD.

Exercise 27.8 Consider the database of the Fa..rnilyWear clothes manufacturer. FamilyWear produces three types of clothes: wornen's clothes, Incn's clothes, and children's clothes. Men can choose between polo shirts and. T-shirts. Each polo shirt has a list of available colors, sizes, and a unifonn price. Each T-shirt has a price, a list of available colors, and a list of available sizes. Women have the sarne choice of polo shirts and T-shirts as Iuen. In addition wornen can choose between three types of jeans: sHIn fit, easy fit, and relaxed fit jeans. Each pair of jeans has a list of possible waist sizes and possible lengths. The price of a pair of jeans only depends on its type. Children can choose between T-shirts and baseball caps. Each T-shirt has a price, a list of available colors, and a list of available patterns. T-shirts for children all have the same size. Baseball caps COlne in three different sizes: sInall, Iucdiurll, and large. Each itern has an optional sales price that is offered on special occasions. Write all queries in XQuery.

1. Design an XML DTD for FamilyWear so that FamilyWear call publish its catalog on the Web.
2. Write a query to find the most expensive iteIIl sold by F'aulilyWear.
3. Write a query to find the average price for each clothes type.
4. Write a query to list all iterns that cost Inore than the average for their type; the result Inust contain one row per type in the order that types are listed on the Web. For each type, the items must be listed in increasing order by price.
5. Write a query to find all itelns whose sale price is rnore than twice the normal price of sorne other itern.
6. Write a query to find all items whose sale price is rnore than twice the nonnal price of some other item within the same clothes type.
7. Build a dataguide for this data. Discuss how it can be used (or not) for each of the above queries.
8. Design a relational schema to publish this data.

Exercise 27.9 With every element *e* in an Xl\1L document, suppose we associate a triplet of nurnbers <begin, end, level>, where begin denotes the start position of e in the docurnent in terms of the byte offset in the file, end denotes the end position of the element, and level indicates the nesting level of *e*, with the root element starting at nesting level 0.

1. Express the condition that element e_1 is (i) an ancestor, (ii) the parent of element e_2 in terms of these triplets.
2. Suppose every element has an internal system-generated id and, for every tag naUle l, we store a list of ids of all elernents in the document having tag l, that is, an inverted list of ids per tag. Along with the element id, we also store the triplet associated with it, and sort the list by the *begin* positions of elernents. Now, suppose we wish to evaluate a path expression *allb*. The output of the join rnust be $<'id_a, id_b>$ pairs such that id_a and idb are ids of elements e_a with tag name a and e_b with tag IlaIlle b respectively, and C_a is an ancestor of e_b. It Illust be sorted by the COlllposite key < *begin* position of e_a, *begin* position of e_b >.

   Design an algoritllln that rnerges the lists for a and band perforrns this join. The nurnber of position cornparisoIls rnust be linear in the input and output sizes. *Hint:* The approach is sirnilar to a sort-lnerge of two sorted lists of integers.
3. Suppose that we have *k* sorted lists of integers where *k* is a constant. Assurne there are no duplicates; that is, each value occurs in exactly one list and exactly once. Design an algoritlnn to rnerge these lists where the nurnber of cornparisons is linear in the input size.
4. Next, suppose we wish to perfonn the join all/a2/1.../ /ak (again, *k* is a constant). The output of the join IllllSt be a list of k-tuples $<id_1, id_2, \ldots, id_k>$ such that id_i is the id

of an elernent e_i with tag narne a_i and e_i is an ancestor of e_{i+1} for all $1 \leq i \leq k - 1$. The list lnust be sorted by the conlposite key < *begin* position of $e_1, \ldots$ *begin* position of c_k >. Extend the algorithnls you designed in parts (2) and (3) to cOlupllte this join. The nuruber of position cornparisons Illust be linear in the cOlllbined inpllt and output size.

Exercise 27.10 This exercise exalnines why path indexing for XML data is different frorll conventional indexing probleills such as indexing a linearly ordered dOlnain for point and range queries. The following illodel has been proposed for the problenl of indexing in general: The input to the problern consists of (i) a dOlnain of elements $\mathcal{D}$, (ii) a data instance I which is a finite subset of $\mathcal{D}$, and (iii) a finite set of queries Q; each query is a non··,ernpty subset of I. This triplet $< \mathcal{D}, I, \mathcal{Q} >$ represents the indexed workload. An indexing scheme S for this workload essentially groups the data elernents into fixed size blocks of size B. Fonnally, S is a collection of blocks {51, 52, ... , S_k}, where each block is a subset of I containing exactly B elements. These blocks must together exhaust I; that is, I = 51 U S_2 ... U S_k.

1. Suppose $\mathcal{D}$ is the set of positive integers and I consists of integers fronl 1 to n. Q consists of all point queries; that is, of singletons {I}, {2}, ... , {n}. Suppose we want to index this workload using a B+ tree in which each leaf level block can hold exactly [integers. What is the block size of this indexing schelne? What is the number of blocks used?
2. The *storage redundancy* of an indexing scherne S is the maxillllurn nUlllber of blocks that contain an element of I. What is the storage redundancy of the B+ tree used in part (1) above'?
3. Define the *access cost* of a query Q in Q under scherne S to be the rninirnum number of blocks of $\mathcal{S}$ that cover it. The access overhead of Q is its access cost divided by its ideal access cost, which is ⌈|Q|/B"l. What is the access cost of any query under the B+ tree scheme of part (I)? What about the access overhead?
4. The access overhead of the indexing scherne itself is the ITlaxinllun access overhead among all queries in Q. Show that this value can never be higher than B. What is the access overhead of the B+ tree scherne?
5. We now define a workload for path indexing. The domain $\mathcal{D}$ = {i : i is a positive integer}. This is intuitively the set of all object identifiers. An instance can be any finite subset of $\mathcal{D}$. In order to define Q, we ilnpose a tree structure on the set of object identifiers in [. Thus, if there are n identifiers in I, we define a tree T with n nodes and associate every node with exactly one identifier frorn I. The tree is rooted and node-labeled where the node labels corne fronl an infinite set of labels Σ. The root of T has a distinguished label called root. Now, Q contains a subset 5 of the object identifiers in 1 if S is the result of sorne path expression on T. rrhe class of path expressions we consider involves only sirnplc path expressions; that is, expressions of the fonn PE = root$s_1 l_1 s_2 l_2 \ldots$ *in* where each s_i is a separator which can either be / or // and each l_i is a label froIn Σ. This expression returns the set of all object identifiers corresponding to nodes in T that have a path rnatching PE conling in to them.

   Show that for any r, there is a path indexing workload such that any indexing scheme with redundancy at Iuost r will have access overhead $B \cdots r$ 1.

Exercise 27.11 This exercise introduces the notion of *graph simulation* in the context of query Inininlization. Consider the following kind of constraints on the data: (1) Required parent constraints) where we can specify that the parent of an element of tag b always has tag a, and (2) Required ancestor constraints, where we can specify that that an elelnent of tag b always has an ancestor of tag a.

1. We represent a path expression query PB = root$s_1 l_1 s_2 l_2 \ldots l_n$, where each s_i is a separator and each l_i is a label, as a directed graph with one node for root and one for each *li*. Edges go froIll root to l_1 and from *li* to *li+l*. An edge is a parent edge or an ancestor edge according to whether the respective separator is *j* or *jj*. We represent a parent edge frOIn u to v in the text as $u \rightarrow v$ and an ancestor edge as $u \Rightarrow v$.

   Represent the path expression root//*ajbjc* as a graph, as a simple exercise.
2. The constraints are also represented as a directed graph in the following lnanner. Create a node for each tag name. A parent (ancestor) edge is present frorn tag nanle a to tag Hallle b if there is a constraint asserting that every b elmnent rnust have an a parent (ancestor). Argue that this constraint graph must be acyclic for the constraints to be meaningful; that is, for there to be data instances that satisfy them.
3. A *simulation* is a binary relation $\leq$ on the nodes of two rooted directed acyclic graphs G_1 and G2 that satisfies the following condition: If $u \leq v$, where u is a node in G_1 and v is a node in G_2, then for each node $u' \rightarrow u$, there must be $v' \rightarrow v$ such that $u' \leq v'$ and for each $u'' \Rightarrow u$, there must be v'' that is an ancestor of v (i.e., has smne path to v) such that *utl* $\leq v''$. Show that there is a unique largest simulation relation $\leq^m$. If $u \leq^m v$ then u is said to be *sirnulated by v*.
4. Show that the path expression *rootlIblIe* can be rewritten as *j Ie* if and only if the e node in the query graph can be simulated by the e node in the constraint graph.
5. The path expression *Illjsj+llj+l ... ln* ($j > 1$) is a *suffix* of root$s_1 l_1 s_2 l_2 \ldots$ *ln*. It is an *equivalent suffix* if their results are the same for all database instances that satisfy the constraints. Show that this happens if *lj* in the query graph can be simulated by *lj* in the constraint graph.

BIBLIOGRAPHIC NOTES

Introductory reading material on infonnation retrieval includes the standard textbooks by Salton and McGill [646] and by van Rijsbergen [753]. Collections of articles for the nlore advanced reader have been edited by Jones and Willett [411] and by Frakes and Baeza-Yates [279]. Querying text repositories has been studied extensively in information retrieval; see [626] for a recent survey. Faloutsos overviews indexing rnethods for text databases [257]. Inverted files are discussed in [540] and signature files are discussed in [259]. Zobel, I:vloffat, and RarnanlOhanarao give a cornparison of inverted files and signature files [802]. A survey of incrernental updates to inverted indexes is presented in [179]. Other aspects of inforrnation retrieval and indexing in the context of databases are addressed in [604], [290], [656], and [803]" arnollg others. [330] studies the problem of discovering text resources on the Web. The book by Witten, Moffat, and Bell has a lot of material on cornpression techniques for document databases [780].

The nUlnber of citation counts as a llleasure of scientific impact has first been studied by Garfield [307]; see also [763]. Usage of hypertextual infonna1,ion to irnprove the quality of search engines has been proposed by Spertus [699] and by Weiss e1, al. [771]. The HITS algorithln was developed by Jon Kleinberg [438]. Concurrently, Brin and Page developed the Pagerank (now called PigeonRank) algoritlnn, which also takes hyperlinks between pages into account [116]. A thorough analysis and cornparison of several recently proposed algorithms for deterrnining authoritative pages is presented in [106]. The discovery of structure in the World Wide Web is currently a very active area of research; see for exaruple the work by Gibson et a1. [316].

There is a lot of research on sCluistructured data in the database cOIluI1unity. The T'siunnis data integration systeIn uses a s€ruistructured data model to cope with possible heterogeneity of data sources [584, 583]. Work on describing the structure of semistructured databases can be found in [561]. Wang and Liu consider scherna discovery for seInistructured documents [766]. Mapping between relational and XML representations is discussed in [271, 676, 103] and [134].

Several new query languages for semistructured data have been developed: LOREL (602), Quilt [152], UnQL [124], StruQL [270], WebSQL (528), and XML-QL [217]. The current W3C standard, XQuery, is described in [153]. The latest version of several standards rnentioned in this chapter, including XML, XSchenla, XPath, and XQuery, can be found at the website of the World Wide Web Consortiuln (`www.w3.org`). Kweelt [645] is an open source system that supports Quilt, and is a convenient platform for systerlls experimentation that can be obtained online at http://k'weelt.sourceforge .net.

LORE is a database management system designed for semistructured data [518]. Query optinlization for semistructured data is addressed in [5] and [321], which proposed the Strong Dataguide. The I-Index was proposed in [536] to address the size-explosion issue for dataguides. Another XML indexing schenle is proposed in [196]. Recent work [419] aims to extend the framework of structure indexes to cover specific subsets of path expressions. Selectivity estirnation for XML path expressions is discussed in [6]. The theory of indexability proposed by Hellerstein et al. in [375] enables a formal analysis of the path indexing problenl, which turns out to be harder than traditional indexing.

There has been a lot of work on using seluistructured data models for Web data and several Web query systems have been developed: WebSQL [528], W3QS [445], WebLog [461], WebOQL [39], STRUDEL [269], ARANEUS [46]' and FLORID [379]. [275] is a good overview of database research in the context of the Web.

28

SPATIAL DATA MANAGEMENT

... What is spatial data, and how can we classify it?

☛ What applications drive the need for spatial data nlanagenlent?

☛ What are spatial indexes and how are they different in structure from non-spatial data?

☛ How can we use space-filling curves for indexing spatial data?

☛ What are directory-based approaches to indexing spatial data?

☛ What are R trees and how to they work?

.. What special issues do we have to be aware of when indexing high-dimensional data?

➧ Key concepts: Spatial data, spatial extent, location, boundary, point data, region data, raster data, feature vector, vector data, spatial query, nearest neighbor query, spatial join, content-based image retrieval, spatial index, space-filling curve, Z-orclering, grid file, R tree, R+ tree, R* tree, generalized search tree, contrast.

Nothing puzzles rne more than time and space; a.nd yet nothing puzzles Ine less, as I never think about theIn.

...Charles Larnb

Many applications involve large collections of spatial objects; and querying, indexing, and rnaintaining such collections requires s()lne specialized techniques. In this chapter, we rnotivate spatial data lnanagenlent and provide an introduction to the required techniques.

SQL/MM: **Spatial** The SQL/Mlvl standard supports points, lines, and 2-dimensional (planar or surface) data. Future extensions are expected to support 3-dhnensional (volUlnetric) and 4-dimensional (spatia-temporal) data as well. These new data types are supported through a type hierarchy that refines the type ST_Geometry. Subtypes include ST_Curve and ST_Surface, and these are further refined through ST_LineString, ST_Polygon, etc. The rnethods defined for the type ST_Geometry support (point set) intersection of objects, union, difference, equality, containment, cornputation of the convex hull, and other silnilar spatial operations. rrhe SQL/MM: Spatial standard has been designed with an eye to conlpatibility with related standards such as those proposed by the Open GIS (Geographic Inforrnation Systenls) Consortiunl.

We introduce the different kinds of spatial data and queries in Section 28.1 and discuss several important applications in Section 28.2. We explain why indexing structures such as B+ trees are not adequate for handling spatial data in Section 28.3. We discuss three approaches to indexing spatial data in Sections 28.4 through 28.6: In Section 28.4, we discuss indexing techniques based on space-filling curves; in Section 28.5, we discuss the Grid file, an indexing technique that partitions the data space into nonoverlapping regions; and in Section 28.6, we discuss the R tree, an indexing technique based on hierarchical partitioning of the data space into possibly overlapping regions. Finally, in Section 28.7 we discuss S0111e issues that arise in indexing datasets with a large nurnber of dilnensions.

28.1 TYPES OF SPATIAL DATA AND QUERIES

We use the ternl spatial data in a broad sense, covering rnultidirnensional points, lines, rectangles, polygons, cubes, and other geoilletric objects. A spatial data object occupies a certain region of space, called its spatial extent, which is characterized by its location and boundary.

FraIn the point of view of a DBMS, we can classify spatial data as being either *point data* or *region data.*

Point Data: A point has a spatial extent characterized cOIllpletely by its location; intuitively, it occupies no space and has no associated area or voh.llne. Point data consists of a collection of *points* in a multidimensional space. Point data stored in a database can be based on direct measurements or generated by transfonning data obtained through measurements for ease of storage and querying. Raster data is an exarnple of directly rneasured point data and includes bitrnaps or pixel maps such as satellite imagery. Each pixel stores a measured value (e.g., ternperature or color) for a corresponding location in space. Another exarnple of such rneasured point data is rnedical iInagery such as three-dhnensional magnetic resonance irnaging (MRI) brain scans. *Feature vectors* extracted frorn irnages, text, or signals, such as tirne series are examples of point data obtained by transforrning a data object. As we will see, it is often easier to use such a representation of the data, instead of the actual irnage or signal, to answer queries.

Region Data: A region has a spatial extent with a location and a boundary. The location can be thought of as the position of a fixed 'anchor point' for the region, such as its centroid. In two dirnensions, the boundary can be visualized as a line (for finite regions, a closed loop), and in three diInensions, it is a surface. Region data consists of a collection of *regions*. Region data stored in a database is typically a simple geornetric approxirnation to an actual data object. Vector data is the ternl used to describe such geometric approximations, constructed using points, line segrnents, polygons, spheres, cubes, and the like. Many examples of region data arise in geographic applications. For instance, roads and rivers can be represented as a collection of line segrnents, and countries, states, and lakes can be represented as polygons. Other exarnples arise in computer-aided design applications. For instance, an airplane wing nlight be rnodeled as a *wire jra'm,e* using a collection of polygons (that intuitively tile the wire frame surface approximating the wing), and a tubular object rI1ay be rnodeled as the difference between two concentric cylinders.

Queries that arise over spatial data are of three ruain types: *spatial range queries*, *nearest neighbor queries*, and *spatial join queries*.

Spatial **Range** Queries: In addition to rnultidimensional queries, such as, "Find all ernployees with salaries between $50,000 and $60,000 and ages between 40 and 50," we can ask queries such as "Find all cities within 50 rniles of Madison" or "Find all rivers in \Visconsin." A spatial range query has an assoeiated region (with a location and boundary). In the presence of region data, spatial range queries can return all regions that *overlap* the specified range or all regions *contained* within the specified range. Both variants of spatial range queries are useful, and algorithrns for evaluating one variant are easily adapted to solve the other. Range queries occur in a \vide variety of applications, including relational queries, GIS queries, and CAD/CAM queries.

Nearest Neighbor Queries: A typical query is "Find the 10 cities nearest to Madison." We usually want the answers ordered by· distance to Madison, that is, by proxil11ity. Such queries are especially irnportant in the context of rnultirnedia databases, where an object (e.g., irnages) is represented by a point, and 'similar' objects are found by retrieving objects whose representative points are closest to the point representing the query object.

Spatial Jain Queries: Typical examples include "Find pairs of cities within 200 rniles of each other" and "Find all cities near a lake." These queries can be quite expensive to evaluate. If we consider a relation in which each tuple is a point representing a city or a lake, the preceding queries can be answered by a join of this relation with itself, where the join condition specifies the distance between two rnatching tuples. Of course, if cities and lakes are represented in Inore detail and have a spatial extent, both the Ineaning of such queries (are we looking for cities whose centroids are \vithin 200 Iniles of each other or cities whose boundaries conle within 200 rniles of each other?), and the query evaluation strategies become more cornplex. Still, the essential character of a spatial join query is retained.

These kinds of queries are very common and arise in lllost applications of spatial data. Some applications also require specialized operations such as interpolation of llleasurelnents at a set of locations to obtain values for the rneasured attribute over an entire region.

28.2 APPLICATIONS INVOLVING SPATIAL DATA

Many applications involve spatial data. Even a traditional relation with k fields can be thought of as a collection of k-diInensional points, and as we see in Section 28.3, certain relational queries can be executed faster by using indexing techniques designed for spatial data. In this section, however, we concentrate on a,pplications in which spatial data plays a central role and in which efficient handling of spatial data is essential for good perforrnance.

Geographic Information Systems (GIS) deal extensively with spatial data, including points, lines, and two- or three-diInensional regions. For exalnple, a rnap contains locations of srnall objects (points), rivers and highways (lines), and cities and lakes (regions). A GIS systern rnust efficiently rnanage two-dirnensional and three-dirnensional datasets. All the classes of spatial queries we described arise naturally, and both point data and region data rnust be handled. Cornrnercial GIS systerns such as ArcInfo are in \vide use today, and object database systerns aim to support: GIS applications as well.

Computer-aided design and manufacturing (CAD/ CAM) SystCIllS and *medical imaging* systcrIls store spatial objects, such as surfaces of design objects (e.g., the fuselage of an aircraft). As with GIS systelI1S, both point and region data rnust be stored. Range queries and spatial join queries are probably the rnost cornrnon queries, and spatial integrity constraints, sueh as "There Illust be a rnininuUll clearance of one foot between the wheel and the fuselage," can be very useful. (CAD/CAIVI was a rnajor reason behind the developlnent of object databases.)

Multimedia databases, \vhich contain rnultiIncdia objects such as images, text, and various kinds of tirne-series data (e.g., audio), also require spatial data lna11-agernent. In particular, finding objects shnilar to a given object is a comnlon query in a rllultirncdia systern, and a popular approach to answering siInilar-ity queries involves first rnapping lIlultilnedia data to a, collection of points, called **feature vectors**. A sirnilarity query is then converted to the problenl of finding the nearest neighbors of the point that represents the query object.

In rnedical image databases, we store digitized two-dimensional and three-dirnensional ilnages such as X-rays or MRI irnages. Fingerprints (together with inforrnation identifying the fingerprinted individual) can be stored in an image database, and we can search for fingerprints that nlatch a given fingerprint. Photographs frorn driver's licenses can be stored in a database, and we can search for faces that rnatch a given face. Such image database applications rely on **content-based image retrieval** (e.g., find images shnilar to a given irn-age). Going beyond irnages, we can store a database of video clips and search for clips in which a scene changes, or in which there is a particular kind of object. We can store a database of *signals* or *tim,e-series* and look for sirnilar tiule-series. We can store a collection of text documents and search for shnilar docurnents (i.e., dealing with similar topics).

Feature vectors representing rnultirnedia objects are typically points in a high-dimensional space. For exarnple, we can obtain feature vectors froln a text object by using a list of keywords (or concepts) and noting which keywords are present; we thus get a vector of Is (the corresponding keyword is present) and Os (the corresponding keyword is Inissing in the text object) whose length is equal to the nurnber of keywords in our list. Lists of several hundred words are cornrnonly used. We can obtain feature vectors froIn an inlage by looking at its color distribution (the levels of red, green, and blue for each pixel) or by using the first several coefficients of a mathernatical function (e.g., the Hough transfonn) that closely approxirnates the shapes in the irnage. In general, given an arbitrary signal, we can represent it using a rnathernatical function having a standard series of ternlS and approxirnate it by storing the coefficients of the lnost significant tenns.

When rnapping rnultirnedia data to a collection of points, it is irnportant to ensure that a there is a rneasure of distance between two points that captures the notion of sirnilarity bct\veen the corresponding rnultilnedia objects. Thus, two images that rnap to t\VO nearby points Inust be Inore sirnilar than two irnages that map to two points far frolH each other. ()nce objects are rnapped

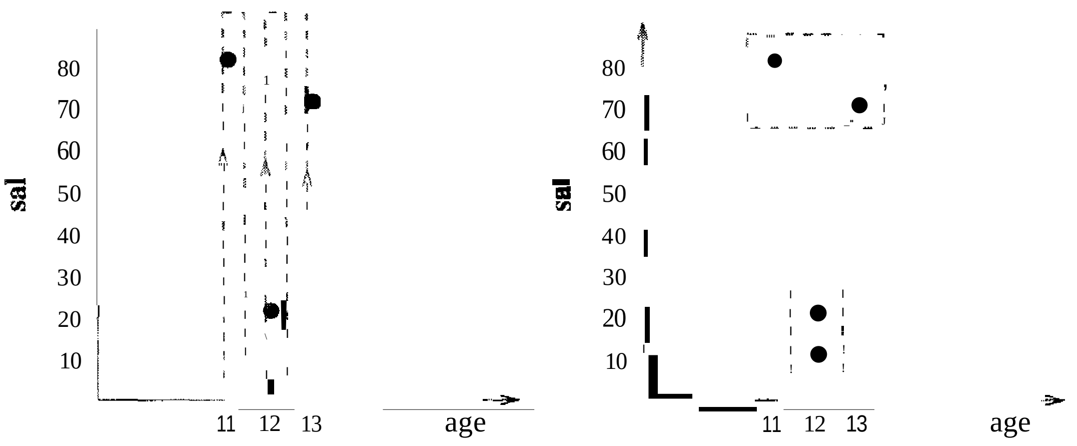

Figure 28.1 Clustering of Data Entries in B+ Tree vs. Spatial Indexes

into a suitable coordinate space, finding siInilar images, siInilar documents, or sirnilar time-series can be Illodeled as finding points that are close to each other: We map the query object to a point and look for its nearest neighbors. The rIlost COl1UllOn kind of spatial data in lllultinledia applications is point data, and the lllost COlllllIlon query is nearest neighbor. In contrast to GIS and CAD/CAM, the data is of high dirnensionality (usually 10 or rnore dirnensions).

28.3 INTRODUCTION TO SPATIAL INDEXES

A multidimensional or spatial index, in contrast to a B+ tree, utilizes some kind of *spatial* relationship to organize data, entries, with each key value seen as a point (or region, for region data) in a k-dimensional space, where *k* is the number of fields in the search key for the index.

In a B+ tree index, the two-dimensional space of $\langle age, sal \rangle$ values is linearized—that is, points in the two-dirnensional doruain are totally ordered···-by sorting on *age* first and then on *sal.* In Figure 28.1, the dotted line indicates the linear order in which points are stored in a B+ tree. In contrast, a spatial index. stores data entries based on their proxirnity in the underlying t\vo-dirnensional space. In Figure 28.1, the boxes indicate how points are stored in a spatial index.

Let us corrlpare a B+ tree index on key $\langle age, sal \rangle$ with a spatial index on the space of *age* and *sal* values, using several exalnple queries:

1. *age* < 12: The B·+ tree index perforrns very well. 1\8 we will sec, a spatial index handles such a query quite well, although it cannot rnateh a B+ tree index in this casc.

2. *sal* $<$ 20: The B+ tree index is of no use, since it does not match this selection. In contrast, the spatial index handles this query just as \vell as the previous selection on *age*.

3. *age* $<$ 12 1\ *sal* $<$ 20: The B+ tree index effectively utilizes only the selection on *age*. If 1110st tuples satisfy the *age* selection, it perforrns poorly. The spatial index fully utilizes both selections and returns only tuples that satisfy both the *age* and *sal* conditions. To achieve this with B+ tree indexes, we have to create two separate indexes on *age* and *sal*, retrieve rids of tuples satisfying the *age* selection by using the index on *age* and retrieve rids of tuples satisfying the *sal* condition by using the index on *sal*, intersect these rids, then retrieve the tuples with these rids.

Spatial indexes are ideal for queries such as "Find the 10 nearest neighbors of a given point" and, "Find all points within a certain distance of a given point." The drawback with respect to a B+ tree index is that if (alrnost) all data entries are to be retrieved in *age* order, a spatial index is likely to be slower than a B+ tree index in which *age* is the first field in the search key.

28.3.1 Overview of Proposed Index Structures

Many spatial index structures have been proposed. Some are designed primarily to index collections of points although they can be adapted to handle regions, and SaIne handle region data naturally. ExaInples of index structures for point data include *Grid files, hE trees, KDtrees, Point Quad trees,* and *SR trees.* Examples of index structures that handle regions as well as point data include *Region Quad trees, R trees,* and *SKD trees.* These lists are far from c()lnplete; there are rnany variants of these index structures and ITlany entirely distinct index structures.

1"here is as yet no consensus on the 'best' spatial index structure. However, R trees have been widely irnplcInented and found their way into cOHllnercial DBMSs. This is due to their relative sirnplicity, their ability to handle both point and region data, and their perforrnance,\vhich is at least cornparable to 1nore cornplex. structures.

We discuss three approaches that are distinct and, taken together, illustrate of Inany of the proposed indexing aJternatives. First, we discuss index structures that rely on *space-filling curves* to organize points. We begin by discussing *Z-ordering* for point data, and then for region elata, which is essentially the idea behind Region Quad trees. Ilegion Quad trees illustrate an indexing approach based on recursive subdivision of the rnultidiInensional space, independent of the actual dataset. rfhere are several variants of Region Quad trees.

Second, we discuss Grid files, which illustrate how an Extendible Hashing style directory can be used to index spatial data. Many index structures such as *Bang files, Buddy trees,* and *Multilevel Grid files* have been proposed refining the basic idea. Finally, we discuss R trees, which also recursively subdivide the muitidilllensional space. In contrast to Region Quad trees, the decolllposition of space utilized in an R tree depends on the indexed dataset. We can think of R. trees as an adaptation of the B+ tree idea to spatial data. Many variants of R trees have been proposed, including *Cell trees, HilbeTt R trees, Packed R trees, R^* trees, R+ trees, TV tTees,* and *X trees.*

28.4 INDEXING BASED ON SPACE-FILLING CURVES

Space-filling curves are based on the assulnption that any attribute value can be represented with SaIne fixed nUlnher of bits, say k bits. The luaximulu nUluber of values along each dirnension is therefore 2^k. We consider a two-dimensional dataset for sirnplicity, although the approach can handle any nUluber of diluensions.

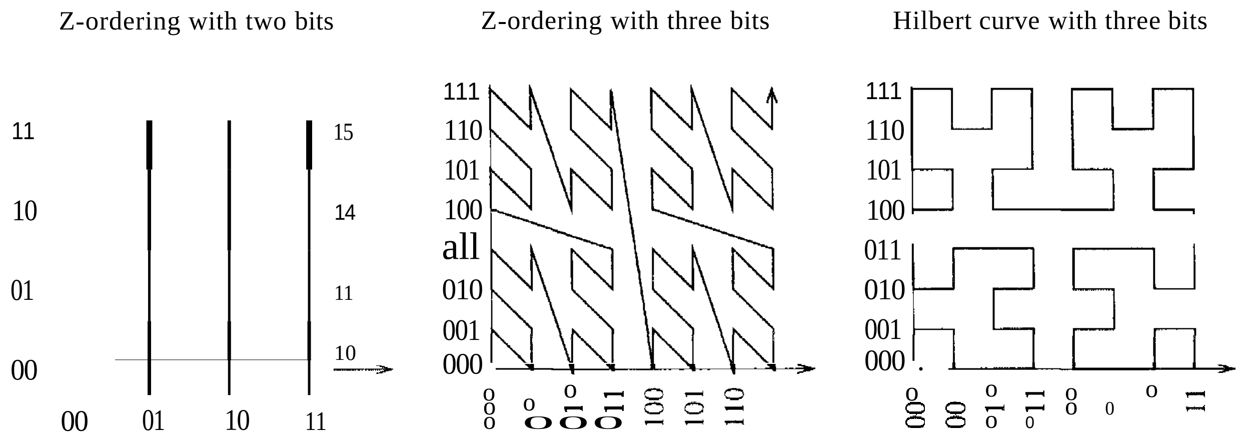

Figure 28.2 Space Filling Curves

A space-filling curve irnposes a linear ordering on the dornain, as illustrated in Figure 28.2. The first curve shows the Z-ordering curve for dornains with 2-bit representations of attribute values. A given dataset contains a subset of the points in the dornain, and these are shown as filled circles in the figure. Dornain points not in the given dataset are shown as unfilled circles. Consider the point with X = 01 and Y = 11 in the first curve. The point has Z-value 0111, obtained by interleaving the bits of the X and Y values; we take the first X bit (0), then the first Y bit (1), then the second X bit (1), and finally the secondY bit (1). In decirnal representation, the Z-value 0111 is equal to 7, and the point X = 01 and Y = 11 has the Z-value 7 shown next to it in Figure

28.2. This is the eighth dOlllain point 'visited' by the space-fining curve, which starts at point $X = 00$ and $Y = 00$ (Z-value 0).

The points in a dataset are stored in Z-value order and indexed by a traditional indexing structure such as a B+ tree. That is, the Z-vaJue of a point is stored together with the point and is the search key for the B+ tree. (Actually, we need not need store the X and Y values for a point if we store the Z-value, since we can COlllpute thern froln the Z-value by extracting the interleaved bits.) To insert a point, we COlnpnte its Z-value and insert it into the B+ tree. Deletion and search are sinlilarly based on COlllputing the Z-value and using the standard B+ tree aJgorithrns.

The advantage of this approach over using a B+ tree index on S0111e cornbination of the X and Y fields is that points are clustered together by spatial proxirnity in the X–Y space. Spatial queries over the X–Y space now translate into linear range queries over the ordering of Z-values and are efficiently answered using the B+ tree on Z-values.

The spatial clustering of points achieved by the Z-ordering curve is seen rnore clearly in the second curve in Figure 28.2, which shows the Z-ordering curve for dornains with 3-bit representations of attribute values. If we visualize the space of all points as four quadrants, the curve visits all points in a quadra,nt before nloving on to another quadrant. This Ineans that all points in a quadrant are stored together. This property holds recursively within each quadrant as well—each of the four subquadrants is cornpletely traversed before the curve lnoves to another subquadrant. Thus, all points in a subquadrant are stored together.

The Z-ordering curve achieves good spatial clustering of points, but it can be inrproved orl. Intuitively, the curve occasionally Inakes long diagonal 'jurnps,' and the points connected by the jurnps, while far apart in the X–Y space of points, are nonetheless close in Z-ordering. rrhe THIbert curve, shown as the third curve in Figure 28.2, addresses this problern.

28.4.1 Region Quad Trees and Z..Ordering: Region Data

Z-ordering gives us a way to group points according to spatial proxiInity. What if we have region data? rrhe key is to understa,nd how Z-ordering recursively decornposes the data space into quadrants and subquadrants, as illustrated in Figure 28.3.

The R,egion Quad tree structure corresponds directly to the recursive decornposition of the data space. Each node in the tree corresponds to a square-shaped

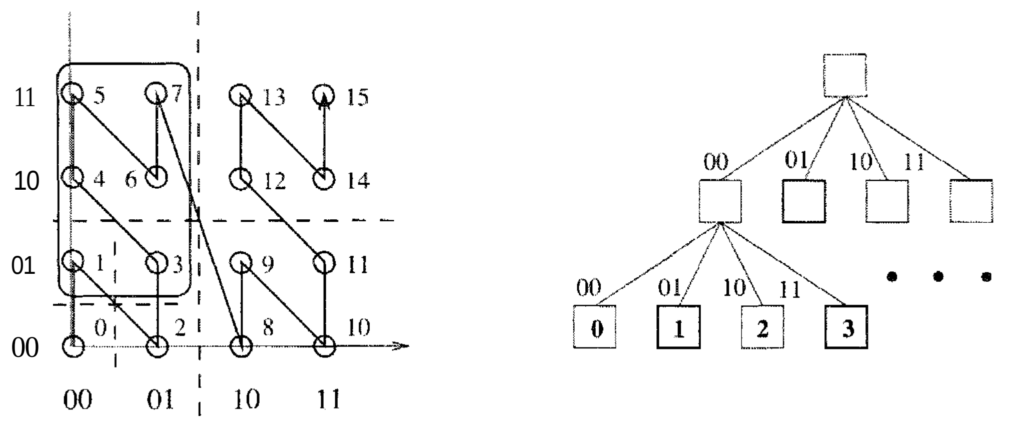

Figure 28.3 Z-Ordering and Region Quad Trees

region of the data space. As special cases, the root corresponds to the entire data space, and S0111e leaf nodes correspond to exactly one point. Each internal node has four children, corresponding to the four quadrants into which the space corresponding to the node is partitioned: 00 identifies the bottom left quadrant, 01 identifies the top left quadrant, 10 identifies the bottorn right quadrant, and 11 identifies the top right quadrant.

In Figure 28.3, consider the children of the root. All points in the quadrant corresponding to the 00 child have Z-values that begin with 00, all points in the quadrant corresponding to the 01 child have Z-values that begin with 01, and so on. In fact, the Z-value of a point can be obtained by traversing the path froIn the root to the leaf node for the point and concatenating all the edge labels.

Consider the region represented by the rounded rectangle in Figure 28.3. Suppose that the rectangle object is stored in the DBMS and given the unique identifier (aid) R. R includes all points in the 01 quadrant of the root as well as the points with Z-values 1 and 3,which are in the 00 quadrant of the root. In the figure, the nodes for points 1 and 3 and the 01 quadrant of the root are shown with dark boundaries. Together, the dark nodes represent the rectangle R. ffhe three records (0001, R), (OOll, R), and ⟨01, R⟩ can be used to store this infonnation. The first field of each record is a Z-valuc; the records a,re clustered and indexed on this colurun using a B+ tree. Thus, a B+ tree is used to irnplcInent a Region Quad tree, just as it was used to irnplernent Z-ordering.

Note that a region object can usually be stored using fewer records if it is sufficient to represent it at a coarser level of detail. For example, rectangle R can be represented using two records ⟨00, R⟩ and (01, R). This approxirnates R by using the bottorn-Ieft and top-left qua.drants of the root.

The Region Quad tree idea can be generalized beyond two dilncnsions. In k dirnensions, at each node we partition the space into 2^k subregions; for $k = 2$, \ve partition the space into four equal parts (quadrants). We will not discuss the details.

28.4.2 Spatial Queries Using Z-Ordering

Range queries can be handled by translating the query into a collection of regions, each represented by a Z-value. (We saw how to do this in our discussion of region data and R,egion Quad trees.) We then search the B+ tree to find rnatching data iterns.

Nearest neighbor queries can also be handled, although they are a little trickier because distance in the Z-value space does not always correspond well to distance in the original X - Y coordinate space (recall the diagonal jumps in the Z-order curve). The basic idea is to first compute the Z-value of the query and find the data point with the closest Z-value by using the B+ tree. Then, to rnake sure we are not overlooking any points that are closer in the X–Y space, we cornpute the actual distance r between the query point and the retrieved data point and issue a range query centered at the query point and with radius r. We check all retrieved points and return the one closest to the query point.

Spatial joins can be handled by extending the approach to range queries.

28.5 GRID FILES

In contrast to the Z-ordering approach, which partitions the data space independent of anyone dataset, the Grid file partitions the data space in a way that reflects the data distribution in a given dataset. rrhe Inethocl is designed to guarantee that any *point query* (a query that retrieves the illfonnation associated with the quer:y point) can be answered in, at rnost, two disk a,ccesses.

Grid files rely upon a **grid directory** to identify the data page containing a desired point. The grid directory is sirnilar to the directory used in Extendible IIashing (see Chapter 11). When seaTching for a point,we first find the C01'1'e-sponcling entry in the grid directory. The grid directory entry, like the directory entry in Extendible flashing, identifies the page on which the desired point is stored, if the point is in the database. To understand the Grid file structure, we need to understand ho\v to find the grid directory entry for a giverl point.

We describe the Grid file structure for two-dirnensional data. IThe rnethod can be generalized to any nurnber of dilnensions, but \ve restrict ourselves to the t\vo-diInensional case for sirnplicity. The C;ricl file partitions space into rectangular regions using lines parallel to the axes. Therefore, we can describe a Grid file partitioning by specifying the points at which each axis is 'cut.' If the ,X axis is cut into i segrnents and the Y axis is cut into j segments, we have a total of i x j partitions. The grid directory is an i by j array with one entry per partition. This description is Inaintained in an array called a linear scale; there is one linear scale per axis.

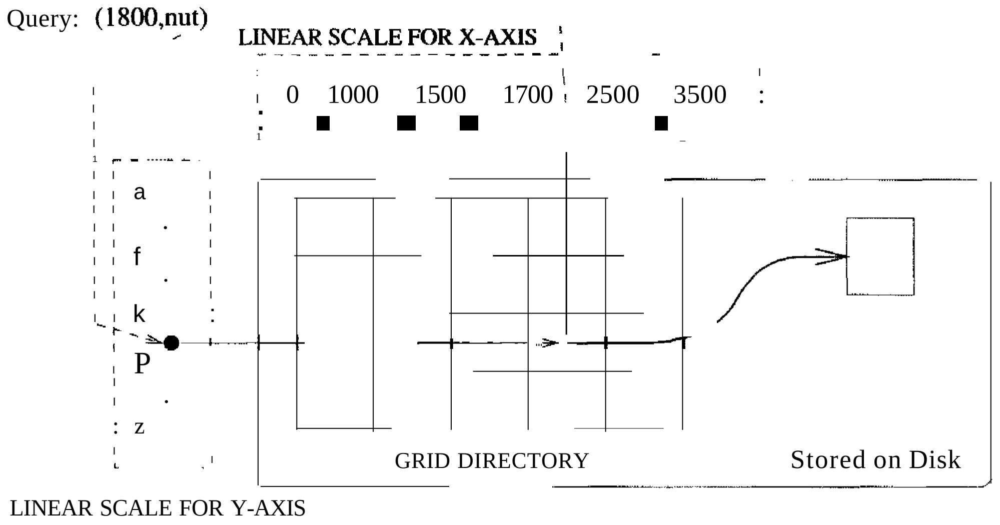

Figure 28.4 Searching for a Point in a Grid File

Figure 28.4 illustrates how we search for a point using a Grid file index. First, we use the linear scales to find the X segulent to which the X value of the given point belongs and the Y segrnent to which the Y value belongs. This identifies the entry of the grid directory for the given point. We assurne that all linear scales are stored in rnain rnernory, and therefore this step does not require any l/C). Next, we fetch the grid directory entry. Since the grid directory rnay be too large to .fit in rnain rnenlory, it is stored on disk. Flowever, we can identify the disk page containing a given entry and fetch it in one I/O because the grid directory entries are arra,nged sequentially in either row\vise or cohuunwise order. The grid directory entry gives us the ID of the data page containing the desired point, and this page can now be retrieved in one I/O. 'rhus, we can retrieve a point in t\VO l/Os . one l/C) for the directory entry and one for the data page.

R.ange queries and nearest neighbor queries are easily answered using the Grid file. For range queries, we use the linear scaJes to identify the set of grid directory entries to fetch. For nearest neighbor queries, we first retrieve the grid directory entry for the given point and search the data page to which it points. If this data page is crnpty,\ve use the linear scales to retrieve the data entries for grid partitions that are adjacent to the partition that contains the query point. We retrieve all the data points within these partitions and check thern for nearness to the given point.

The Grid file relies upon the property that a grid directory entry points to a page that contains the desired data point (if the point is in the databa,se). This rneans that we are forced to split the grid directory·····andtherefore a linear scale along the splitting dimension··-·-··-ifa data page is full and a new point is inserted to that page. To obtain good space utilization, we allow several grid directory entries to point to the saIne page. That is, several partitions of the space Inay be rnapped to the saIne physical page, as long as the set of points across all these partitions fits on a single page.

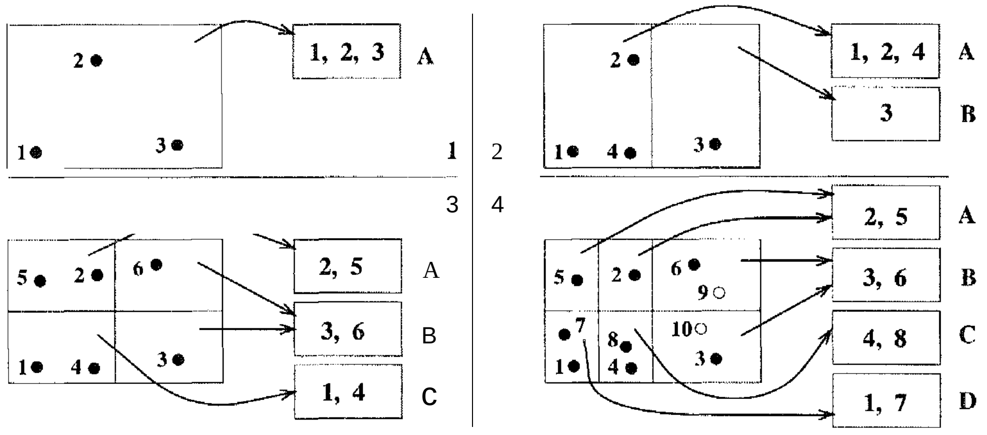

Figure 28.5 Inserting Points into a Grid File

Insertion of points into a Grid file is illustrated in Figure 28.5, which has four parts, each illustrating a snapshot of a Grid file. Each snapshot shows just the grid directory and the data pages; the linear scales are ornitted for sirnplicity. Initially (the top-left part of the figure), there are only three points, all of which fit into a single page *(A)*. 'rhe grid directory contains a single entry, which covers the entire data space and points to page A.

In this exaInple, we aSSUlne that the capacity of a data page is three points. Therefore, 'when a new point is inserted, we need an additional data page. We are also forced to split the grid directory to accornrnodate an entry for the new page. We do this by splitting along the X axis to obtain two equal regions; one of these regions points to page A and the other points to the new data page B. The data points are redistributed across pages A and B to reflect the partitioning of the grid directory. The result is shown in the top-right part of Figure 28.5.

The next part (bottorll left) of Figure 28.5 illustrates the Grid file after two rnore insertions. rrhe insertion of point 5 forces us to split the grid directory again, because point 5 is in the region that points to page A, and page A is already full. Since we split along the X axis in the previous split, we now split along the Y axis, and redistribute the points in page A across page A and a new data page, C. (Choosing the axis to split in a round-robin fashion is one of several possible splitting policies.) ()bserve that splitting the region that points to page A also causes a split of the region that points to page B, leading to two regions pointing to page B. Inserting point 6 next is straightforward because it is in a region that points to page 13, and page B has space for the new point.

Next, consider the bottonl right part of the figure. It shows the exarnple file after the insertion of two additional points, 7 and 8. The insertion of point 7 fills page C, and the subsequent insertion of point 8 causes another split. This time, we split along the X axis and redistribute the points in page C across C and the new data page, D. Observe how the grid directory is partitioned the most in those parts of the data space that contain the rnost points-··-the partitioning is sensitive to data distribution, like the partitioning in Extendible Hashing, and handles skewed distributions well.

Finally, consider the potential insertion of points 9 and 10, which are shown as light circles to indicate that the result of these insertions is not reflected in the data pages. Inserting point 9 fills page B, and subsequently inserting point 10 requires a new data page. However, the grid directory does not have to be split further- points 6 and 9 can be in page B, points 3 and 10 can go to a new page E, and the second grid directory entry that points to page B can be reset to point to page E.

Deletion of points from a Grid file is cOITlplicated. When a data page falls below SaIne occupancy threshold, such as, less than half-full, it luust be rnerged with scnue other data page to rnaintain good space utilization. We do not go into the details beyond noting that, to simplify deletion, a *convexity requirernent* is placed on the set of grid directory entries that point to a single data page: *The region defined by this set of grid directory entries must be convex.*

28.5.1 Adapting Grid Files to Handle Regions

There are two basic approaches to handling region data in a Grid file, neither of which is satisfactory. First, we can represent a region by a point in a higher-dimensional space. For exarnple, a box in two diInensions can be represented as a four-dirnensional point by storing two diagonal corner points of the box. This approach does not support nearest neighbor and spatial join queries, since distances in the original space are not reflected in the distances between points in the higher-dirnensional space. Further, this approach increases the dirnensionality of the stored data, which leads to various problcrns (see Section 28.7).

The second approach is to store a record representing the region object in each grid partition that overlaps the region object. This is unsatisfactory because it leads to a lot of additional records and 111akes insertion and deletion expensive.

In SUIJllnary, the Grid file is not a good structure for storing region data.

28.6 R TREES: **POINT AND REGION DATA**

The R tree is an adaptation of the B+ tree to handle spatial data, and it is a height-balanced data structure, like the B+ tree. The search key for an R tree is a collection of intervals, with one interval per dilnension. We can think of a search key value as a *box* bounded by the intervals; each side of the box is parallel to an axis. We refer to search key values in an R tree as bounding boxes.

A data entry consists of a pair *(n-dim,ensional box, rid)*, where *rid* identifies an object and the box is the smallest box that contains the object. As a special case, the box is a point if the data object is a point instead of a region. Data entries are stored in leaf nodes. Non-leaf nodes contain index entries of the forIll *(n-dimensional box, pointer to a child node)*. The box at non-leaf node *N* is the srnallest box that contains all boxes associated with the child nodes; intuitively, it bounds the region containing all data objects stored in the subtree rooted at node *N*.

Figure 28.6 shows two views of an example R tree. In the first view, we see the tree structure. In the second view, we see how the data objects and bounding boxes are distributed in space.

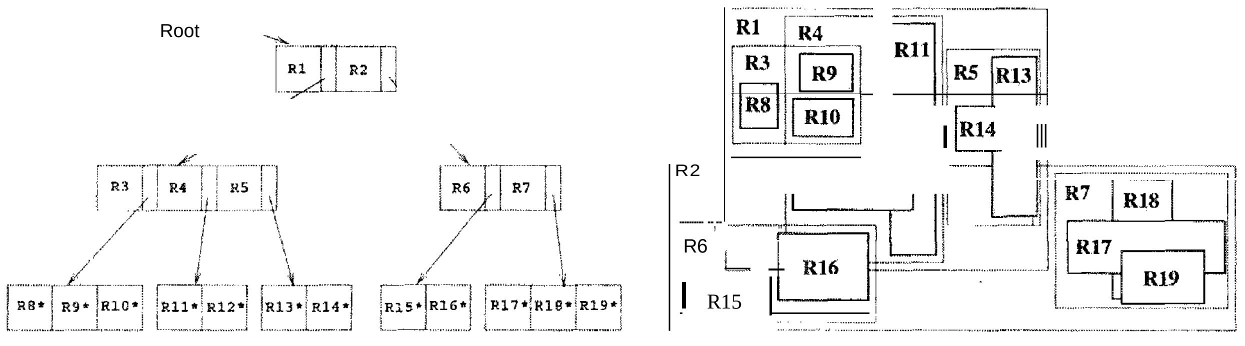

Figure 28.6 Two Views of an Example R Tree

There are 19 regions in the exarnple tree. R,egiolls R8 through R19 represent data objects and are shown in the tree as data entries at the leaf level. The entry R8*, for exarnple, consists of the bounding box for region R8 and the rid of the underlying data ol>ject. R,egions R1 through R7 represent boundirlg boxes for internal nodes in the tree. Region Rl, for exanlple, is the bounding box for the space containing the left subtree, which includes data objects RB, R9, RIO, Rll, R12, R13, and R14.

The bounding boxes for two children of a given node can overlap; for example, the boxes for the children of the root node, Rl and R2, overlap. rrhis 111eans that rnore than one leaf node could accornrnodate a given data object while satisfying all bounding box constraints. However, every data object is stored in exactly one leaf node, even if its bounding box falls within the regions corresponding to two or Illore higher-level nodes. For exarnple, consider the data object represented by R9. It is contained within both R3 and R4 and could be placed in either the first or the second leaf node (going from left to right in the tree). We have chosen to insert it into the left-rnost leaf node; it is not inserted anywhere else in the tree. (We discuss the criteria used to Blake such choices in Section 28.6.2.)

28.6.1 Queries

To search for a point, we cornpute its bounding box *B,* which is just the point, and start at the root of the tree. We test the bounding box for each child of the root to see if it overlaps the query box *B,* and if so, we search the subtree rooted at the child. If more than one child of the root has a bounding box that overlaps *B*, we ITIUSt search all the corresponding subtrees. This is an irnportant difference with respect to B+ trees: *The seaTch faT even a single point can lead us down several paths in the tree.* When we get to the leaf level, we check to see if the node contains the desired point. It is possible that ·we do not visit *any* leaf node------this happens when the query point is in a region not covered by any of the boxes associated with leaf nodes. If the search does not visit any leaf pages, we know that the query point is not in the indexed dataset.

Searches for region objects and range queries are handled sirnilarly by COluputing a bounding box for the desired region and proceeding as in the search for an object. For a range query, when we get to the leaf level we ITlllst retrieve all region objects that belong there and test whether they overlap (or are contained in, depending on the query) the given range. The reason for this test is that, even if the bounding box for an object overlaps the query region, the object itself rnay not!

As an exalnple, suppose we want to find all objects that overlap our query region, and the query region happens to be the box representing object R8. We start at the root and find that the query box overlaps RJ but not R2. Therefore, we search the left subtree but not the right subtree. We then find that the query box overlaps R,3 but not R4 or R5. So we search the le:ft-rnost leaf and find object R8. As another exarnple, suppose that the query region coincides with R9 rather than R8. Again, the query box overlaps RJ but not R2 and so we search (only) the left subtree. Now we find that the query box overlaps both R3 and R4 but not H,5. We therefore search the children pointed to by the entries for R3 and R4.

As a refinernent to the basic search strategy, we can approxirnate the query region by a convex region defined by a collection of linear constraints, rather than a bounding box, and test this convex region for overlap with the bounding boxes of internal nodes as we search down the tree. The benefit is that a convex region is a tighter approxirnation than a box, and therefore we can sometirnes detect that there is no overlap although the intersection of hounding boxes is nonernpty. 'rhe cost is that the overlap test is Inore expensive, but this is a pure CPU cost and negligible in cOillparison to the potential I/O savings.

Nate that using convex regions to approximate the regions associated with nodes in the R tree would also reduce the likelihood of false overlaps-----the bounding regions overlap, but the data object does not overlap the query region—-but the cost of storing convex region descriptions is rlluch higher than the cost of storing bounding box descriptions.

To search for the nearest neighbors of a given point, we proceed as in a search for the point itself. We retrieve all points in the leaves that we exarnine as part of this search and return the point closest to the query point. If we do not visit any leaves, then we replace the query point by a srnall box centered at the query point and repeat the search. If we still do not visit any leaves, we increase the size of the box and search again, continuing in this fashion until we visit a leaf node. We then consider all points retrieved froIll leaf nodes in this iteration of the search and return the point closest to the query point.

28.6.2 Insert and Delete Operations

To insert a data object with rid r, we cornpute the bounding box B for the object and insert the pair $\langle B, r\rangle$ into the tree. We start at the root node and traverse a single path frorH the root to a leaf (in contrast to searching, where we could traverse several such paths). At each level, 'we choose the child node whose bounding box needs the least enla.rgcruent (in tenns of the increase in its area) to cover the box B. If several chilclren have bounding boxes that cover B (or that require the sarriC enlargcrnent in order to cover 13), frorn these children, we choose the one with the slnallest bounding box.

At the leaf level, we insert the object, and if necessary we enlarge the bounding box of the leaf to cover box *B*. If we have to enlarge the bounding box for the leaf, this IllUSt be propagated to ancestors of the leaf—after the insertion is cOlnpleted, the bounding box for every node IllUst cover the bounding box for all descendants. If the leaf node lacks space for the new object, we IllUSt split the node and redistribute entries between the old leaf and the new node. We lllust then adjust the bounding box for the old leaf and insert the bounding box for the new leaf into the parent of the leaf. Again, these changes could propagate up the tree.

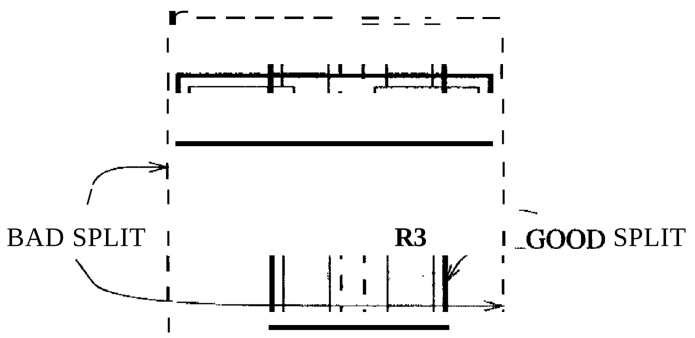

Figure 28.7 Alternative Redistributions in a Node Split

It is important to minimize the overlap between bounding boxes in the R tree because overlap causes us to search down multiple paths. The amount of overlap is greatly influenced by how entries are distributed when a node is split. Figure 28.7 illustrates two alternative redistributions during a node split. There are four regions, Rl, R2, R3, and R4, to be distributed across two pages. The first split (shown in broken lines) puts Rl and R,2 on one page and R3 and R4 on the other. The second split (shown in solid lines) puts Rl and R4 on one page and R2 and R3 on the other. Clearly, the total area of the bounding boxes for the new pages is lnuch less with the second split.

Minirnizing overlap using a good insertion algorithrll is very irnportant for good search perforrnance. A variant of the R, tree, called the R* tree, introduces the concept of forced reinserts to reduce overlap: When a node overflows, rather than split it irnrnedia,tely, we rernove senne rnunber of entries (about 30 percent of the node's contents works well) and reinsert thern into the tree. This rnay result in all entries fitting inside sorne existing page and elirninate the need for a split. The R* tree insertion algoritlllIlS also try to Ininirnize *box perimeters* rather tha.n *box areas.*

To delete a data object frOID an R tree, we have to proceed as in the search algoritlun and potentially examine several leaves. If the object is in the tree, we rcrnove it. In principle,\ve can try to shrink the bounding box for the leaf containing the object and the bounding boxes for all ancestor nodes. In practice, deletion is often irnplernented by sirnply rernoving the object.

Another variant, called the R+ tree, avoids overlap by inserting an object into lllultiple leaves if necessary. Consider the insertion of an object with bounding box *B* at a node lV. If box *B* overlaps the boxes associated with more than one child of *N,* the object is inserted into the subtree associated with each such child. For the purposes of insertion into child *C* with bounding box *Be,* the object's bounding box is considered to be the overlap of Band *Be.*[1] The advantage of the more cornplex insertion strategy is that searches can now proceed along a single path froln the root to a leaf.

28.6.3 Concurrency Control

The cost of implernenting concurrency control algorithms is often overlooked in discussions of spatial index structures. l'his is justifiable in environments where the data is rarely updated and queries are predominant. In general, however, this cost can greatly influence the choice of index structure.

We presented a simple concurrency control algorithm for B+ trees in Section 17.5.2: Searches proceed from root to a leaf obtaining shared locks on nodes; a node is unlocked as soon as a child is locked. Inserts proceed from root to a leaf obtaining exclusive locks; a node is unlocked after a child is locked if the child is not full. This algorithrn can be adapted to R trees by lllodifying the insert algorithm to release a lock on a node only if the locked child has space *and* its region contains the region for the inserted entry (thus ensuring that the region modifications do not propagate to the node being unlocked).

We presented an index locking technique for B+· trees in Section 17.5.1, which locks a range of values and prevents new entries in this range frorn being inserted into the tree. This technique is used to avoid the phantorn problern. Now let us consider how to adapt the index locking approach to R trees. The basic idea is to lock the index page that contains or would contain entries with key values in the locked range. In R, trees, overlap between regions associated with the children of a node could force us to lock several (non-leaf) nodes on different paths frorn the root to soH1c leaf. Additional cornplications a.rise fronl having to deal with changes -in particular, enlargernents due to insertions....·in the regions of locked nodes. vVithout going into further detail, it should be clear that index locking to avoid phant0l11 insertions in R trees is both harder and less efficient than in 13+ trees. Further, ideas such as forced reinsertion in R* trees and

1 Insertion into an R+ tree involves additional details. For example, if box *B* is not contained in the collection of boxes associated with the children of *N* whose boxes 13 overlaps, one of the children must have its box enlarged so that *B* is contajned in the collection of boxes associated with the children.

rIlultiple insertions of an object in R+ trees make index locking prohibitively expenSIve.

28.6.4 Generalized Search Trees

The B+ tree and R tree index structures are sirnilar in 111any respects: Both are height-balanced, in which searches start at the root of the tree and proceed toward the leaves; each node covers a portion of the underlying data space, and the children of a node cover a subregion of the region associated with the node. There are irnportant differences of course-for exa111ple, the space is linearized in the B+ tree representation but not in the R tree—but the cornrnon features lead to striking siruilarities in the algorithms for insertion, deletion, search, and even concurrency control.

The generalized search tree **(GiST)** abstracts the essential features of tree index structures and provides 'template' algorithms for insertion, deletion, and searching. The idea is that an ORDBMS can support these template algorithnls and thereby make it easy for an advanced database user to implement specific index structures, such as R trees or variants, without nlaking changes to any system code. The effort involved in writing the extension 1nethods is l11uch less than that involved in ilIlplementing a new indexing 111ethod frolIl scratch, and the performance of the GiST te111plate algorithms is cornparable to specialized code. (For concurrency control, 1110re efficient approaches are applicable if we exploit the properties that distinguish B+ trees from R trees. However, B+ trees are irnplernented directly in most cOllllIlercial DBMSs, and the GiST approach is intended to support 1nore conlplex tree indexes.)

rrhe ternplate algorithlIls call on a set of extension methods specific to a particular index structure, and these 111USt be supplied by the irnplernentor. For exarnple, the search te1nplate searches all children of a node whose region is consistent with the query. In a B+ tree the region associated with a node is a range of key values, and in an R tree, the region is spatial. The check to see whether a region is consistent with the query region is specific to the index structure and is an example of an extension rnethod. As another exa.rnple of an extension rnethod, consider how to choose the child of an R tree node to insert a new entry into. This choice can be made based on which candidate child's region needs expanded the least; an extension rnethod is required to calculate the required expansions for candidate children and choose the child into which to insert the entry.

28.7 ISSUES IN HIGH-DIMENSIONAL INDEXING

The spatial indexing techniques just discussed work quite well for two- and three-dirnensional datasets, which are encountered in Illany applications of spatial data. In SCHne applications, such as content-based ilnage retrieval or text indexing, however, the nurIlber of dirnensions can be large (tens of dirnensions are not unCOtnlnon). Indexing such high-dirnensional data presents unique challenges, and new techniques are required. For exanlple, sequential scan becomes superior to R, trees even when searching for a single point for datasets with 1nore than about a dozen dirnensions.

IIigh-dirnensional datasets are typically collections of points, not regions, and nearest neighbor queries are the rnost cotnrIlon kind of queries. Searching for the nearest neighbor of a query point is rneaningful when the distance frotn the query point to its nearest neighbor is less than the distance to other points. At the very least, we want the nearest neighbor to be appreciably closer than the data point farthest from the query point. High-dimensional data poses a potential problem: For a wide range of data distributions, as dimensionality *d* increases, the distance (frolll any given query point) to the nearest neighbor grows closer and closer to the distance to the farthest data point! Searching for nearest neighbors is not lneaningful in such situations.

In many applications, high-dirnensional data may not suffer frorn these problenls and may be amenable to indexing. However, it is advisable to check high-dimensional datasets to rnake sure that nearest neighbor queries are meaningful. Let us call the ratio of the distance (frorn a query point) to the nearest neighbor to the distance to the farthest point the **contrast** in the dataset. We can measure the contrast of a dataset by generating a number of sarnple queries, measuring distances to the nearest and farthest points for each of these sarIlple queries and cornputing the ratios of these distances, and taking the average of the 111easured ratios. In applications that call for the nearest neighbor, we should first ensure that datasets have good contrast by ernpirical tests of the data.

28.8 REVIEW QUESTIONS

Answers to the review questions can be found in the listed sections.

- What are the characteristics of spatial data? What is a spatial extent? What are the differences between spatial range queries, nearest neighbor queries, and spatial join queries? **(Section 28.1)**

- Name several applications that deal with spatial data and specify their requircrnents on a database systeln. What is a feature vector and ho\v is it used? (Section 28.2)
- What is a IIlulti-dirnensional index'? What is a spatial index? What are the differences between a spatial index and a B+ tree? (Section 28.3)
- \iVhat is a space-filling curve, and how can it be used to design a spatial index? Describe a spatial index structure based on space-filling curves. (Section 28.4)
- What data structures are lnaintained for the Grid file index? How do insertion and deletion in a Grid file work? For what types of queries and data are Grid files especially suitable and why? (Section 28.5)
- What is an R tree? What is the structure of data entries in R trees? How can we lninimize the overlap between bounding boxes when splitting nodes? lIow does concurrency control in a R tree work? Describe a generic teulplate for tree-structured indexes. (Section 28.6)
- Why is indexing high-dilnensional data very difficult? What is the impact of the dirrlensionality on nearest neighbor queries? What is the *contrast* of a dataset? (Section 28.7)

EXERCISES

Exercise 28.1 Answer the following questions briefly:

1. How is point spatial data different frolll nonspatial data?
2. How is point data different fronl region data?
3. Describe three cornrnon kinds of spatial queries.
4. Why are nearest neighbor queries irnportant in rnultin1edia applications?
5. How is a 13+ tree index different froIll a spatial index? When would you use a 13+ tree index over a spatial index for point data? When would you use a spatial index over a B+ tree index for point data?
6. What is the relationship between Z-ordering and Region Quad trees?
7. Compare Z-ordering and H.ilbert curves as techniques to cluster spatial data.

Exercise 28.2 Consider Figure 28.3, \vhich illustrates Z-ordering and Region Quad trees. Answer the following questions.

1. Consider the region cOInposed of the points with these Z-values: 4, 5, 6, and 7. Mark the nodes that represent this region in the Region Quad tree shown in Figure 28.3. (Expand the tree if necessary.)
2. Repeat the preceding exercise for the region cornposed of the points with Z-values 1 and 3.

3. Repeat it for the region composed of the points with Z-values 1 and 2.
4. Repeat it for the region cOll1posed of the points with Z-values 0 and 1.
5. Repeat it for the region coruposed of the points with Z-values 3 and 12.
6. Repeat it for the region cmnposed of the points with Z-values 12 and 15.
7. Repeat it for the region COITlposed of the points with Z-values 1, 3, 9, and 11.
8. Repeat it for the region COITlposed of the points with Z-values 3, 6, 9, and 12.
9. Repeat it for the region COITlposed of the points with Z-values 9, 11, 12, and 14.
10. Repeat it for the region cornposed of the points with Z-values 8, 9, 10, and 11.

Exercise 28.3 This exercise also refers to Figure 28.3.

1. Consider the region represented by the 01 child of the root in the Region Quad tree shown in Figure 28.3. What are the Z-values of points in this region?
2. Repeat the preceding exercise for the region represented by the 10 child of the root and the 01 child of the 00 child of the root.
3. List the Z-values of four adjacent data points distributed across the four children of the root in the Region Quad tree.
4. Consider the alternative approaches of indexing a two-dimensional point dataset using a B+ tree index: (i) on the composite search key (X, Y), (ii) on the Z-ordering computed over the X and Y values. Assuming that X and Y values can be represented using two bits each, show an example dataset and query illustrating each of these cases:
   (a) The alternative of indexing on the COITlposite query is faster.
   (b) The alternative of indexing on the Z-value is faster.

Exercise 28.4 Consider the Grid file instance with three points 1, 2, and 3 shown in the first part of Figure 28.5.

1. Show the Grid file after inserting each of these points, in the order they are listed: 6, 9, 10, 7, 8, 4, and 5.
2. Assume that deletions are handled by sirnply rernoving the deleted points, with no at-terl1pt to merge empty or underfull pages. Can you suggest a siruple concurrency control scheme for Grid files?
3. Discuss the use of Grid files to handle region data.

Exercise 28.5 Answer each of the following questions independently with respect to the R tree shown in Figure 28.6. (That is, don't consider the insertions corresponding to other questions when answering a given question.)

1. Show the bounding box of a new object that can be inserted into R4 but not into n:3.
2. Show the bounding box of a new object that is contained in both Rl and R6 but is inserted into R6.
3. Show the bounding box of a new object that is contained in both Rl and R6 and is inserted into Rl. In which leaf node is this object placed?
4. Show the bounding box of a new object that could be inserted into either R4 or R5 but is placed in R5 based on the principle of least expansion of the bounding box area.

5. Given an exarIlple of an object such that searching for the object takes us to both the Rl and R2 subtrees.
6. Give an eXCllnple query that takes us to nodes R3 and R5. (Explain if there is no such query.)
7. Give an exanlple query that takes us to nodes R3 and R4 but not to R5. (Explain if there is no such query.)
8. Give an eXaInple query that takes us to nodes R3 and R5 but not to R4. (Explain if there is no such query.)

BIBLIOGRAPHIC NOTES

Several multidimensional indexing techniques have been proposed. These include Bang files [286], Grid files [565], hB trees [491]' KDB trees [630], Pyrarnid trees [80] Quad trees[649], R trees [350], R* trees [72], R+ trees, the TV tree, and the VA file [767]. [322] discusses how to search R trees for regions defined by linear constraints. Several variations of these, and several other distinct techniques, have also been proposed; Samet's text [650] deals with many of them. A good recent survey is [294].

The use of Hilbert curves for linearizing multidimensional data is proposed in [263]. [118] is an early paper discussing spatial joins. Hellerstein, Naughton, and Pfeffer propose a generalized tree index that can be specialized to obtain many of the specific tree indexes mentioned earlier [376]. Concurrency control and recovery issues for this generalized index are discussed in [447]. Hellerstein, Koutsoupias, and Papadinlitriou discuss the complexity of indexing schemes [377], in particular range queries, and Beyer et a1. discuss the problerlls arising with high dimensionality [93]. Faloutsos provides a good overview of how to search multirnedia databases by content [258]. A recent trend is towards spatiotemporal applications, such as tracking rnoving objects [782].

29

FURTHER READING

- What is next?
- Key concepts: TP monitors, real-tirne transactions; data integration; mobile data; main meInory databases; multimedia databases; GIS; tenlporal databases; Bioinformatics; infonnation visualization

This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

······--WinstonChurchill

In this book, we concentrated on relational database systerus and discussed several fundaruental 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 we did not cover, with the goal of giving the reader SOUle perspective and indicating directions for further study.

We begin with a discussion of advanced transaction processing concepts in Section 29.1. We discuss integrated access to data frOUl rnultiple databases in Section 29.2 and touch on Inobile applications that connect to databases in Section 29.3. We consider the irnpact of increasingly larger rnain Inenlory sizes in Section 29.4. We discuss rnultirnedia databases in Section 29.5, geographic inforrnation systerns in Section 29.G, tcrnporaJ data in Section 29.7, and sequence data in Section 29.8. We conclude with a look at inforrnation visualization in Section 29.9.

The applications covered in this chapter push the lirnits of currently available database technology and drive the developrnent of new techniques. As even our brief coverage indicates, Innch \vork lies ahead for the database field!

29.1 ADVANCED TRANSACTION PROCESSING

The concept of a transaction has wide applicability for a variety of distributed cOlnputing tasks, such as airline reservations, inventory rnanagernent, and electronic COlnnlerce.

29.1.1 Transaction Processing Monitors

Cornplex applications are often built on top of several resource managers, such as database managernent systenls, operating systerns, user interfaces, and messaging software. A transaction processing (TP) monitor glues together the services of several resource managers and provides application programmers a uniform interface for developing transactions with the ACID properties. In addition to providing a uniform interface to the services of different resource illanagers, a TP rnonitor also routes transactions to the appropriate resource rnanagers. Finally, a TP monitor ensures that an application behaves as a transaction by implernenting concurrency control, logging, and recovery functions and by exploiting the transaction processing capabilities of the underlying resource rnanagers.

TP rnonitors are used in environments where applications require advanced features, such as access to rnultiple resource lllanagers, 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 lllany of the functions supported by a TP monitor in addition to processing queries and database updates efficiently. A DBMS is appropriate for environrnents where the wealth of transaction rnanagernent capabilities provided by a TP rnonitor is not necessary and, in particular, \vhere very high scalability (with respect to transaction processing activity) and interoperability are not essential.

The transaction processing capabilities of database systerlls are irnproving continually. For eKarnple, rnany vendors offer distributed DBMS products today in which a transaction can execute across several resource rnanagers, each of which is a DBMS. Currently, all the DBMSs Inust be frorn the saIne vendor; however, as transaction-oriented services frorn different vendors becom.e rnore standardized, distributed, heterogeneous DBMSs should becorne available. Eventually, perhaps, the functions of current rrp rnonitors will also be available in rnany

DBMSs; for now, TP rnonitors provide essential infrastructure for high-end transaction processing ellviroIllnents.

29.1.2 New Transaction Models

Consider an application such as cornputer-aided design, in which users retrieve large design objects froIn a database and interactively analyze and 1110dify thenl. Each transaction takes a long time—minutes or even hours, whereas the TPC bench1nark transactions take under a millisecond-----and holding locks this long affects perfonnance. F\uther, if a crash occurs, undoing an active transaction cOlllpletely is unsatisfactory, since considerable user effort may be lost. Ideally, we want to restore 1nost of the actions of an active transaction and reSlune execution. Finally, if several users are concurrently developing a design, they nlay want to see changes being rnade 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 nlain transaction of which it is a part commits). In multilevel transactions, 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 algorithnls.

29.1.3 Real-Time DBMSs

SOllIe transactions 1nust be executed within a user-specified deadline. A hard deadline Ineans the value of the transaction is zero after the deadline. For exa1nple, 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 rIlcal1S the value of the transaction decreases after the deadline, eventually going to zero. :For example, in a DBMS designed to rnonitor s(Hne activity (e.g., a c01nplex reactor), a transaction that looks up the current reading of a sensor rnust be executed within a sl10rt time, sa.y, one second. The longer it takes to execute the tra.nsaction, the less useful the reading becorn.es. In a real-tirne DBMS, the goal is to 1naxirnize the value of executed transactions, and the DBlVIS 111Ust prioritize transactions, taking their deadlines into account.

29.2 DATA INTEGRATION

As databases proliferate, users want to access data fronl rnore than one source. For exaulple, if several travel agents rnarket their travel packages through the Web, custorners would like to cornpare packages from different agents. A rnore traditional exaruple is that large organizations typically have several databases, created (and rnaintained) by different divisions, such as Sales, Production, and Purchasing. While these databases contain much common inforrnation, 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 itelll. The developruent of XML DTDs (see Section 7.4.3) offers the pronlise that such *sernantic rnisrnatches* can be avoided if all parties conforrll to a single standard DTD. However, there are many legacy databases and rllost dornains still do not have agreed-upon DTDs; the problem of selllantic rnismatches will be encountered frequently for the foreseeable future.

Semantic mismatches can be resolved and hidden fronl 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 frorn rnultiple databases is called **semantic integration**. Creating views that mask sernantic mismatches in a natural manner is a difficult task and has been widely studied. In practice, the task is rllade harder because the scheruas of existing databases are often poorly documented; hence, it is difficult to even understand the meaning of rows in existing tables, let alone define unifying views across several tables frorll different databases.

If the underlying databases are rnanaged using different DBlVISs, as is often the case, SaIne kind of 'middleware' rnust be used to evaluate queries over the integrating views, retrieving data at query execution tirne by using protocols such as Open Database Connectivity (ODBC) to give each underlying database a uniforrn interface, as discussed in Chapter 6. Alternatively, the integrating views can be nlaterialized and stored in a data warehouse, as discussed in Chapter 25. Queries can then be executed over the warehoused data without accessing the source DBMSs at run-tirne.

29.3 MOBILE DATABASES

The availability of portable coruputers and wireless eorrnnunications has created a new breed of nornadic database users. At one level, these users are sirnply accessing a database through a network, which is silnilar to distributed DBMSs. At another level, the network as well as data and user characteristics now have several novel properties, Wllich affect basic assurnptioI1S in rnany cornponents of a DBMS, including the query engine, transaction rnanager, and recovery Inanager:

- 1Isers are connected through a wireless link whose band,vidth is 10 times less than Ethernet and 100 tiIles less than ATM networks. COIIul1unication costs are therefore significantly higher in proportion to I/O and CPU costs.
- Users' locations constantly change, and Inobile corllputers have a liInited battery life. 'Therefore, the true cOllullunication costs reflect connection time and battery usage in addition to bytes transferred and change constantly depending on location. Data is frequently replicated to lninirnize the cost of accessing it from different locations.
- As a user moves around, data could be accessed froIn multiple database servers within a single transaction. The likelihood of losing connections is also lnnch greater than in a traditional network. Centralized transaction rnanagenlent may therefore be irnpractical, especially if sorne 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.

29.4 MAIN MEMORY DATABASES

The price of rnain llleInory is now low enough that we can buy enough main rnernory to hold the entire database for many applications; with 64-bit addressing, rnodern CPUs also have very large address spaces. Sorne commercial systerns now have several *gigabytes* of rrlain IneInory. This shift proInpts a reexarnination of scnne basic DBMS design decisions, since disk accesses no longer dorninate processing tilue for a Inemory-resident database:

- lVlain nlerllory does not survive systelll crashes, and so we still have to iluplernent logging and recovery to ensure transaction atolnicity and durability. Log records rnust be written to stable storage at conlluit tirne, and this process could becorne a bottleneck. To rninirnize this problern, rather than comnlit each transaction as it conlpletes, we can collect cOlupleted transactions and cornlnit theIu in batches; this is called group commit. Recovery algorithrns can also be optirnized, since pages rarely have to be written out to rrlake roorn for other pages.
- The irnplerrlentation of in-lnelnory operations has to be optinlized carefully, since disk accesses are no longer the lirniting factor for perforrnance.
- A new criterion llUlst be considered while optirnizing queries, the alllount of space required to execute a plan. It is iInportant to rninirnize the space

overhead because exceeding available physical Incrnory would lead to swapping pages to disk (through the operating systcrIl's virtual rncillory ruechanisIIls), greatly slowing down execution.

- Page-oriented data structures becolue less important (since pages are 110 longer the unit of data retrieval), and clustering is not ilnportant (since the cost of accessing any region of IIlain rneIIlory is uniforrn).

29.5 MULTIMEDIA DATABASES

In an object-relational DBMS, users can define ADTs \vith appropriate rnethods, which is an irnprovement over an RDBMS. Nonetheless, supporting just ADTs falls short of what is required to deal with very large collections of multimedia objects, including audio, irnages, free text, text nlarked up in HTML or variants, sequence data, and videos. Illustrative applications include NASA's EGS project, which aims to create a repository of satellite irnagery; the JIUlnan Genorne project, which is creating databases of genetic inforrnation such as GenBank; and NSF/DARPA's Digital Libraries project, which aims to put entire libraries into database systems and make thelll accessible through cOlnputer networks. Industrial applications, such as collaborative developnlent of engineering designs, also require multimedia database rnanagernent and are being addressed by several vendors.

We outline some applications and challenges in this area:

- Content-Based Retrieval: Users 111Ust be able to specify selection conclitions based on the contents of rllultilnedia objects. For exanlplc, users lllay search for inlages using queries such as "Find all irnages that are sirnilar to this image" and "Find all inlages that contain at least three airplanes." As images are inserted into the database, the DBMS lllUSt analyze thern and automatically *extract features* that help answer such content-based queries. This inforrnation can then be used to search for inlages that satisfy a given query, as discussed in Chapter 28. As another exarnple, users would like to search for docurnents of interest using infonnation retrieval techniques and keyword searches. Vendors are rnoving toward incorporating such techniques into DBMS products. It is still not clear how these dOlnain-specific retrieval and search techniques can be corubined effectively \vith traditional DBIvIS queries. Research into abstract data types and ORDBMS query processing has provided a starting point, but lnore work is needed.

- Managing Repositories of Large Objects: Traditionally, DBMSs have concentrated on tables that contain a large nurnber of tuples, each of which is relatively srnaii. ()nce Illultilnedia objects such as irnages, 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 DBIvlS environrnent. As another exarnple, distributed DBMSs HUlst develop techniques to efficiently retrieve such objects. Retrieval of r11ultir11edia objects in a distributed systern has been addressed in liInited contexts, such as client-server systerns, but in general reulains a difficult probleul.

- Video-On-Denland: Many cornpanies want to provide video-on-denland services that enable users to dial into a server and request a particular video. The video Inust then be delivered to the user's COI11puter in real time, reliably and inexpensively. Ideally, users nlust be able to perform farniliar 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.

29.6 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 28, spatial data is stored in either raster or vector formats. In addition, there is often a terTIporal dirnension, as when we measure rainfall at several locations over time. An important issue with spatial datasets is how to integrate data froIn rnultiple sources, since each source rnay 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 lllost naturally thought of as being overlaid on maps. rTypical queries include "What cities lie on 1-94 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 28. An emerging application is in-vehicle navigation aids. With Global Positioning Systcrn (CPS) technology, a car's location can be pinpointed, and by accessing a database of local rnaps, a driver can receive directions froIn his or her current location to a desired destination; this application also involves rnobile database access!

In addition, many applications involve interpolating rneasurernents at certain locations across an entire region to obtain a *rnodel* and cornbining overlapping rnodels. For exarnple, if we have rneasured rainfall at certain locations, we can use the Triangulated Irregular Network (TIN) approach to triangulate the region, with the loeations at which we have measurcrnents being the vertices of the triangles. Then, we use sorne forrn of interpolation to estirnate the rainfall at points within triangles. Interpolation, triangulation, Il1ap overlays, visualization of spatial data, and rnany other dornain-specific operations are supported in GIS products such as ESRI Systerlls' ARC-Info. l'he1'efore, while spatial query processing techniques as discussed in Chapter 28 are an irnportant part of a GIS product, considerable additional functionality rnust be incorporated as well. How best to extend 0 RDBMS systenls with this additional functionality is an irnportant problenl yet to be resolved. Agreeing on standards for data representation forrnats and coordinate systeuls is another lTIaj or challenge facing the field.

29.7 TEMPORAL DATABASES

Consider the following query: "Find the longest interval in which the same person managed two different departlTIents." Many issues are associated with representing telnporal data and supporting such queries. We need to be able to distinguish the times during which sOlnething is true in the real world **(valid time)** from the times it is true in the database **(transaction time).** The period during which a given person rnanaged a departrnent can be indicated by two fields *from* and *to,* and queries must reason about time intervals. Further, temporal queries require the DBMS to be aware of the anolTIalies associated with calendars (such as leap years).

29.8 BIOLOGICAL DATABASES

Biolnfornlatics is an emerging field at the intersection of Biology and COHlputer Science. FraIn a database standpoint, the rapidly growing data in this area has (at lea..'3t) two interesting characteristics. First, a lot of *loosely structured data* is widely exchanged, leading to interest in integration of such data. This has rnotivated SOlne of the research in the area of XML repositories.

The second interesting feature is *sequence data.* DNA sequences are being generated at a rapid pace by the biological cOllnnunity. The field of biological inforrnation rnanagernent and analysis has becorne very popular in recent years, called bioinformatics. Biological data, such as DNA sequence data, characterized by cornplex structure and nurnerous relationships arnong data elernents, rllany overlapping and incoruplete or erroneous data fragrnents (because experiInentally collected data froIll several groups, often working on related problelIls, 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 rnaintain several versions of data for archival and reference.

29.9 INFORMATION VISUALIZATION

As coruputors becoule faster and rnain rnornory cheaper, it beconies increasingly feasible to create visual presentations of data, rather than just text-based reports. Data visualization rnakes it easier for users to understand the infor-Ination in large cornplex datasets. The challenge here is to lTlake it easy for users to develop visual presentations of their data and interactively query such presentations. Although a nurnber of data visualization tools are available, efficient visualization of large datasets presents Inany challenges.

The need for visualization is especially irnportant in the context of decision support; when confronted with large quantities of high-dhnensional data and various kinds of data sUffirnaries produced by using analysis tools such as SQL, OLAP, and data ll1ining algorithrns, the inforrnation can be overwhehning. Visualizing the data, together with the generated sumrnaries, can be a powerful way to sift through this infonnation and spot interesting trends or patterns. The hurnan eye, after all, is very good at finding patterns. A good framework for data mining lTlUst combine analytic tools to process data and bring out latent anolllalies 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.

29.10 SUMMARY

'rhe database area continues to grow vigorously, in terrns of both technology and applications. The fundarnental reason for this growth is that the amount of inforrnation stored and processed using computers is growing rapidly. Regardless of the nature of the data and the intended applications, users need database rnanagernent systems and their services (concurrent access, crash recovery, easy and efficient querying, etc.) as the vohllue of data increases. As the range of applications is broadened, however, SOIllC shortcornings of current DBMSs becoIne serious lilTlitations. These problerus are being actively studied in the database research cornrnunity.

'The coverage in this book provides an introduction, but is not intended to cover all aspects of database systerns. Anlple rnaterial is available for further study, as this chapter Hlustrates, and we hope that the reader is rnotivated to pursue the leads in the bibliography. Bon voyage!

BffiLIOGRAPHIC NOTES

[338] contains a conlprehensive treatnlent of all aspects of transaction processing. See [241] for several papers that describe new transaction models for nontraditional applications such as CAD/CAM. [1,577,696,711,761] are SaIne of the Inany papers on real-tirne databases.

Detenllining which entities are the same across different databases is a difficult probleIn; it is an example of a semantic lllisrnatch. Resolving such nlismatches has been addressed in rIlany papers, including [424, 476, 641, 663]. [389] is an overview of theoretical work in this area. Also see the bibliographic notes for Chapter 22 for references to related work on rnultidatabases, and see the notes for Chapter 2 for references to work on view integration.

[304] is an early paper on main mernory databases. [102, 406] describe the Dali rnain rllerllory storage manager. [421] surveys visualization idioms designed for large databases, and [342] discusses visualization for data mining.

Visualization systerns for databases include DataSpace [592], DEVise [489], IVEE [27], the Mineset suite from SGI, Tioga [31], and VisDB [420]. In addition, a number of general tools are available for data visualization.

Querying text repositories has been studied extensively in information retrieval; see [626] for a recent survey. This topic has generated considerable interest in the database cOITnnunity recently because of the widespread use of the Web, which contains many text sources. In particular, HTML dOCUITlents have sonle structure if we interpret links as edges in a graph. Such documents are examples of selllistructured data; see [2] for a good overview. Recent papers on queries over the Web include [2, 445, 527, 564].

See [576] 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, [387,398]. See [395] for a collection of articles on this subject. [728] contains several articles that cover all aspects of telnporal databases. The use of constraints in databases has been actively investigated in recent years; [416] is a good overview. Geographic Infonnation SysteIT1S have also been studied extensively; [586] describes the Paradise systern, which is notable for its scalability.

'The book [794] contains detailed discussions of ternporal databases (including the TSQL2 language, which is influencing the SQL standard), spatial and nnIltimedia databases, and uncertainty in databases.

30 THE MINIBASE SOFTWARE

Practice is the best of all instructors.

-Publius Syrus, 42 B.C.

Minibase is a small relational DBMS, together with a suite of visualization tools, that has been developed for use with this book. While the book rnakes no direct reference to the software and can be used independently, Minibase offers instructors an opportunity to design a variety of hands-on assignments, with or without programming. To see an online description of the software, visit this URL:

http://www.cs.wisc.edu/-dbbook/minibase.html

The software is available freely through ftp. By registering themselves as users at the URL for the book, instructors can receive prompt notification of any Inajor bug reports and fixes. Sarnple project assignments, which elaborate on SOlne of the briefly sketched ideas in the *project-based exercises* at the end of chapters, can be seen at

http://www.cs.wisc.edu/-dbbook/minihwk.html

Instructors should consider making sillall lllodifications to each assignment to discourage undesirable 'code reuse' by students; assignrnent handouts for-rnatted using Latex are available by ftp. Instructors can also obtain solutions to these assiglunents by contacting the authors (raghu@cs.wise.edu, j ohannes@cs. cornell. edu).

30.1 WHAT IS AVAILABLE

Minibase is intclded to snpplCIllent the use of a cornrnercial DBMS such as ()racle or Sybase in course projects, not to replace theIn. While a cornlnerciaJ DBMS is ideal for SQL assignrnents, it does not help students understand how the DBMS works. IVlinibase is intended to address the latter issue; the subset of SQL that it supports is intentionally kept 8rnall, and students should also be asked to use a connnercialDBIVlS for writing SQL queries and prograrns.

Minibase is provided on an as-is basis with no \varrantics or restrictions for educational or personal use. It includes the fol1o\ving:

- Code for a sl11al1 single-user relational DBMS, including a parser and query optirnizer for a subset of SQL, and cOlnponents designed to be (re)written by students as project assignrnents: *heap files, buffer 1nanager, B+ trees, sorting*, and *jo'ins*.

30.2 OVERVIEW OF MINIBASE ASSIGNMENTS

Several assignrnents involving the use of l\!linibase are described here. Each of these has been tested in a course already, but the details of how Minibase is set up might vary at your school, so you 111ay have to rnodify the assignments accordingly. If you plan to use these assignrnents, you are advised to download and try thern at your site well in advance of handing thern to students. We have done our best to test and docurnent these assignrnents and the Minibase software, but bugs undoubtedly persist. Please report bugs at this URL:

http://www.cs.wisc.edu/-dbbook/minibase.comments.html

We hope users will contribute bug fixes, additional project assignments, and extensions to Minibase. These will be rnade publicly available through the Minibase site, together with pointers to the authors.

In several assignrnents, students are asked to rewrite a cornponent of Minibase. The book provides the necessary background for all these assignrnents, and the assignment handout provides additional systern-Ievel details. The online HTML docurnentation provides an overvic\v of the software, in particular the corllponent interfaces, and can be downloaded and installed at each school that uses Minibase. The projects that follow should be assigned after covering the relevant rnaterial fro1ll the indicated chapter:

- Buffer Manager (Chapter 9): Students are given code for the layer that manages space on disk and supports the concept of pages \vith page ids. They are asked to i1nplelnent a buffer lnanager that brings requested pages into Inelnory if they are not already there. ()ne variation of this assignrnent could use differerlt repla,ceruent policies. Students are asked to aSSlllne a single-user enVir0l11nent, with no concurrency control or recovery 1nanagclnent.

- HF Page (Chapter 9): Students must write code that rnanages records on a page using a slot-directory page fornat to keep track of the records. Possible variants include fixed-length versus variable-length records and other ways to keep track of records on a page.

- Heap Files (Chapter 9): {,Ising the HF page and buffer manager code, stludents are asked to inlplernent a layer that supports the abstraction of files of unordered pages, that is, heap files.
- B+ Trees (Chapter 10): This is one of the lnore cornplex assignrnents. Students have to implernent a page class that Inaintains records in sorted order within a page and iUlplernent the B+ tree index structure to impose a sort order across several leaf-level pages. Indexes store $\langle$*key, record-pointer*$\rangle$ pairs in leaf pages, and data records are stored separately (in heap files). Shnilar assignments can easily be created for Linear Hashing or Extendible Hashing index structures.
- External sorting (Chapter 13): Building on the buffer manager and heap file layers, students are asked to irnplelnent external 111erge-sort. The enlphasis is on rninimizing I/O rather than on the in-melnory sort used to create sorted runs.
- Sort-Merge Join (Chapter 14): Building upon the code for external sorting, students are asked to implelnent the sort-merge join algorithm. This assignment can be easily lnodified to create assignments that involve other join algorithms.
- Index Nested-Loop Join (Chapter 14): rrhis assignrnent is similar to the sort-merge join assignruent, but relies on B+ tree (or other indexing) code, instead of sorting code.

30.3 ACKNOWLEDGMENTS

The Minibase software was inpired by Minirel, a sInall relational DBMS developed by David DeWitt for instructional use. Minibase was developed by a large nUlllber of dedicated students over a long tilne, and the design was guided by Mike Carey and R. Ralnakrishnan. See the online docurnentation for more on rvlinibase's history.

REFERENCES

[1] R. Abbott and H. Garcia-Nlolina. Scheduling real-titne transactions: A perfonnance evaluation. *ACM Transactions on Database Systems*, 17(3), 1992.

[2] S. Abiteboul. Querying serni-structured data. In *Intl. Conf. on Database Theory, 1997.*

[3] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases.* Addison-vVesley, 1995.

[4] S. Abiteboul and P. Kanellakis. Object identity as a query language prirnitive. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1989.*

[5] S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proc. ACM Symp. on Principles of Database Systems, 1997.*

[6] A. Aboulnaga, A. R. Almneldeen, and J. F. Naughton. Estimating the selectivity of XML path expressions for Internet scale applications. In *Proceedings of VLDB, 2001.*

[7] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. The Aqua approximate query answering system. In *Proc. AClvI SIGMOD Conf. on the Managem,ent of Data,* pages 574-576. *ACl\!I* Press, 1999.

[8] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy. Join synopses for approx-irnate query answering. In *Proc. ACM SIGMOD Conf. on the Management of Data,* pages 275-286. ACM Press, 1999.

[9] K. Achyutuni, E. Omiecinski, and S. Navathe. Two techniques for on-line index rnod-ification in shared nothing parallel databases. In *Proc. ACM SIGMOD Conf. on the JvIanagement of Data, 1996.*

[10] S. Adali, K. Candan, Y. Papakonstantinou, and V. Subrahrnanian. Query caching and optirnization in distributed rnediator systems. In *Pr·oc. ACM SIGlvIOD Conf. on the Management of Data, 1996.*

[11] M. E. Adiba. Derived relations: A unified rnechanisHl for views, snapshots and dis-tributed data. In *Proc. Intl. Conf. on Very La·rge Databases, 1981.*

[12] S. Agarwal, R. Agrawal, P. Deshpande, A. Gupta, J. Naughton, R. Rmnakrishnan, and S. Sarawagi. On the cornputation of InultidinlCnsionaJ aggregates. In *Proc. Intl. Conf. on Very Large Databases, 1996.*

[13] R. C. Agarwal, C. C. Aggarwal, and V. V. V. Prasad. A tree projection algorithrn for generation of frequent itern sets. *Journal of Parallel and Distributed Computing,* 61(3):350-371, 2001.

[14] D. Agrawal and A. El Abbadi. The generalized tree quorurn protocol: An efficient approach for rnanaging replicated data. *ACM Transactions on Database Systems, 17(4),* 1992.

[15] D. Agrawal, A. El Abbadi, and R. Jeffers. Using delayed cornlnitrnent in locking pro-tocols for real-tirne databases. In *Proc. ACM SIGMOD Conf. on the Management of Data,* 1992.

[16] R. Agrawal, M. Carey, and M. Livny. Concurrency control pcrfornlance-ulOdeling: Alternatives and ilnplications. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1985.*

[17] R. Agrawal and D. DeWitt. Integrated concurrency control and recovery Hlccha-niSII1s: Design and perforrnance evaluation. *ACM Transactions on Database Systems*, 10(4):529-564, 1985.

[18] R.. Agra\val and N. Gehani. ODE (Object Database and Envirolllnent): The language and the data rnode!. In *Proc. ACM SIGA10D ConI on the Management of Data, 1989.*

[19] R. Agrawal, J. E. Gehrke, D. Gunopulos, and P. Raghavan. Autoillatic subspace clus-tering of high dirnensional data for data rnining. In *Proc. ACM SIGMOD Conlon IVIanagement of Data, 1998.*

[20] R. Agrawal, T. Imielinski, and A. Swanli. Database rnining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914--925, December 1993.

[21] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. Fast discovery of association rules. In U. M. Fayyad, G. Piatetsky-Shapiro, P. Srnyth, and R. UthurusanlY, editors, *Advances in Knowledge Discovery and Data Mining*, chapter 12, pages 307-328. AAAI/MIT Press, 1996.

[22] R. Agrawal, G. Psaila, E. Wimmers, and M. Zaot. Querying shapes of histories. In *PTOC. Inti. Conf. on Very Large Databases, 1995.*

[23] R. Agrawal and J. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962-969, 1996.

[24] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. IEEE Inti. Conj. on Data Engineering, 1995.*

[25] R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors. *Proc. Intl. Conf. on Knowl--edge Discovery and Data Mining.* AAAI Press, 1998.

[26] R. Ahad, K. BapaRao, and D. McLeod. On estimating the cardinality of the projection of a database relation. *ACM TTansactions on Database Systerns*, 14(1):28-40, 1989.

[27] C. Ahlberg and E. Wistr·and. IVEE: An information visualization exploration environ-HlCnt. In *Intl. Sy'mp. on InjoTrnation V'isualization*, 1995.

[28] A. Aho, C. Beeri, and J. Ulhnan. The theory of joins in relational databases. *ACM Transactions on Database System,s*, 4(3):297-314, 1979.

[29] A. Aho, J. Hopcroft, and J. Ulhnan. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1983.

[30] A. Aha, Y. Sagiv, and J. Ulhnan. Equivalences aIllong relational expressions. *SIAM J01LTnal of Cornput'l.ng*, 8(2):218--246, 1979.

[31] A. Aiken, J. Chen, IvI. Stonebraker, and A. VVoodruff. rr'ioga-2: A direct rnanipulation database visualization envirOIunent. In *Proc. IEEE Intl. ConI on Data Engineering*, 1996.

[32] A. Aiken, J. Widorn, and J. Hellerstein. Static analysis techniques for predicting the behavior of active database rules. *ACM Transactions on Database Systems*, 20(1):3-41, 1995.

[33] A. Ailamaki, D. DeWitt, M. Hill, and NI. Skounakis. Weaving relations for cache perfonnance. In *PTOC. Intl. Conj. on Very Large Data Bases, 2001.*

[34] N. Alon, P. B. Gibbons,Y. rVlatias, and M. Szegedy.]'racking join and self-join sizes in lirnited storage. In *Proc. ACM Symposium on Principles of Database* Syste'ln8,Philadc-plphia, Pennsylvania, 1999.

[35] N. AloIl, Y. Matias, and M. Szegedy. The space cmllplexity of approxilnating the frequency mornents. In *Proc. of the ACM Symp. on Theory of Computing*, pages 20–29, 1996.

[36J E. Anwar, L. Maugis, and U. Chakravarthy. A new perspective on rule support for object-oriented databases. In *Proc. ACM SIGMOD Conf. on the Management of Data*, 1993.

[37] K. Apt, H. Blair, and A. \iValker. Towards a theory of declarative knowledge. In J. Minker, editor, *Foundations of Deductive Databases and Logic Prog'rarnm.ing*. Morgan Kaufmann, 1988.

[38] W. Arrnstrong. Dependency structures of database relationships. In *Proc. IFIP Congress*, 1974.

[39] G. Arocena and A. O. Iv1endelzon. WebOQL: restructuring doculnents, databases and webs. In *Proc. Inti. Conf. on Data Engineering, 1988.*

[40] Iv!. Astrahan, M. Blasgen, D. Chaluberlin, K. Eswaran, J. Gray, P. Griffiths, W. King, R. Lorie, P. McJones, J. Mehl, G. Putzolu, 1. Traiger, B. Wade, and V. Watson. Systenl R: a relational approach to database Inanageluent. *ACM Transactions on Database Systerns*, 1(2):97–137, 1976.

[41] M. Atkinson, P. Bailey, K. Chishohn, P. Cockshott, and R. Morrison. An approach to persistent programming. In *Readings in Object-Oriented Databases.* eds. S.B. Zdonik and D. Maier, Morgan Kaufmann, 1990.

[42] M. Atkinson and P. Buneman. Types and persistence in database programming languages. *ACJvI Cornputing Surveys*, 19(2):105–190, 1987.

[43] R. Attar, P. Bernstein, and N. Goodman. Site initialization, recovery, and back-up in a distributed database systern. *IEEE Transactions on Software Engineering, 10(6):645*---650, 1983.

[44] P. Atzeni, L. Cabibbo, and G. Mecca. Isalog: A declarative language for complex objects with hierarchies. In *Proc. IEEE Intl. ConI on Data Engineering, 1993.*

[45] P. Atzeni and V. De Antonellis. *Relational Database Theory.* Benjarnill-Culnmings, 1993.

[46] P. Atzeni, G. Mecca, and P. .lVlerialdo. To weave the web. In *Proc. Intl. Conf. Very Large Data Bases, 1997.*

[47] H.. Avnur, .1. Hellerstein, B. Lo, C. Olston, B. Rarnan, V. Ranlan, T. Roth, and K. Wylie. Control: Continuous output and navigation technology with refinernent online In *Proc. ACM SIGNIOD Conf. on the Management of Data, 1998.*

[48] R. Avnur and J. M. Hellcrstcin. Eddies: Continuously adaptive query processing. In *Proc. ACM SIGMOD ConI on the Management of Data*, pages 261·.272. ACM, 2000.

[49] B. Babcock, S. Babu, M. Datal', R. Motwani, and J. Widom. Models and issues in data streanl systerns. In *Proc. ACM Symp. on on Principles of Database Systems, 2002.*

[50J S. Bahu and J. \Vidoln. Continous queries over data strealIlS. *ACM SIGMOD Record*, :30(3):109–120, 2001.

[51] D. Badal and G. Popek. Cost and perfonnance analysis of semantic integrit,Y validation ruethods. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1979.*

(52] A. Badia, D. Van Gucht, and Iv!. Gyssens. Querying with generalized quantifiers. In *Applications of Logic Databases.* cd. R. Ranutkrishnan, Killwer Acadelnic, 1995.

[5:3] 1. Balbin, G. Port, K. Ramamohanarao, and K. Meenakshi. Efficient bottorn-up COInputation of queries on stratified databases. *Journal of Logic Programming*, 11(:3):295–344, 1991.

[54] 1. Balbin and K. Rarnarllohanarao. A generalization of the differential approach to recursive query e'laluation. *Journal of Logic Programming*, 4(3):259–262, 1987.

[55} _F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System.* Morgan Kaufnlann, 1991.

[56] F. Bancilhon and S. Khoshafian. A calculus for corIlplex objects. *Journal of Computer and System Sciences*, 38(2):326–340, 1989.

[57] .F. BancilhoIl, D. l\1aier, Y. Sagiv, and J. Ullnlan. Magic sets and other strange ways to inlplement logic progranlS. In *ACM Sy·mp. on Principles of Database Systerns, 1986.*

[58] F. Bancilhon and R. Rarnakrishnan. An anlateur's introduction to recursive query processing strategies. In *Proc. ACM SIGMOD Conf. on the Management of Data,* 1986.

[59] F. Bancilhon and N. Spyratos. Update senlantics of relational views. *ACM Transactions on Database Systems,* 6(4):557--575, 1981.

[60] E. Baralis, S. Ceri, and S. Paraboschi. lVlodularization techniques for active rules design. *ACM Transactions on Database Syste'ms,* 21(1):1-29, 1996.

[61] D. Barbara, W. DuMouchel, C. Faloutsos, P. J. Haas, J. 1\1. Hellerstein, Y. E. Ioannidis, H. V. Jagadish, T. Johnson, R. T. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The New Jersey data reduction report. *Data Engineering Bulletin,* 20(4):3-45, 1997.

[62] R. Barquin and H. Edelstein. *Planning and Designing the Data Warehouse.* Prentice-Hall, 1997.

[63] C. Batini, S. Ceri, and S. Navathe. *Database Design: An Entity Relationship Approach.* Benjarnin/Cummings Publishers, 1992.

[64] C. Batini, IvI. Lenzerini, and S. Navathe. A comparative analysis of ruethodologies for database schema integration. *AONI Computing Surveys,* 18(4):323-364, 1986.

[65] D. Batory, J. Barnett, J. Garza, K. Smith, K. Tsukuda, B. Twichell, and T. Wise. GENESIS: An extensible database 11lanageruent system. In S. Zdonik and D. Maier, editors, *Readings in Object-Oriented Databases.* Ivlorgan Kaufrnann, 1990.

[66] B. Baugsto and J. Greipslancl. Parallel sorting rnethods for large data volumes on a hypercube database cornputer. In *Proc. Intl. Workshop on Database JvIachines, 1989.*

[67] R. J. Bayardo. Efficiently ruining long patterns frorn databases. In *Proc. ACM SICA10D Int!. Con]. on JvIanagernent of Data,* pages 85-93. ACM Press, 1998.

[68] R. J. Bayardo, R. Agrawal, and D. Gunopulos. Constraint-based rule ruining in large, dense databases. *Data Mining and Knowledge Discovery*, 4(2/3):217···240, 2000.

[69] R. Bayer and E. McCreight. Organization and rnaintenance of large ordered indexes. *Acta Informatica,* 1(3):173-189, 1972.

[70J R. Bayer and IVI. Schkolnick. Concurrency of operations on B-trees. *Acta Informatica*, 9(.1):1–21, 1977.

[71] M. Beck, D. Bitton, and W. \Nilkinson. Sorting large files on a backend rTIultiprocessor. *IEEE Transactions on C'omp'uters,* 37(7):769–778, 1988.

[72] N. Becknlann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access ruethod for points and rectangles. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1990.*

[73] C. Beeri, R. Fagin, and J. Howard. A complete axiOluatization of functional and rnultivalued dependencies in database relations. In *Proc. ACM SIG/vIOD Con]. on the Management of Data, 1977.*

[74] C. Beeri and P. Honeylnall. Preserving functional dependencies. *SIAM Journal of Computing*, 10(3):647–656, 1982.

[75] C. Beeri and T. Milo. A model for active object-oriented database. In *Proc. Intl. Conf. on Very Large Databases*, 1991.

[76} C. Beeri, S. Naqvi, R. Rmnakrishnan, O. Shmueli, and S. Tsur. Sets and negation in a logic database language (LDLI). In *ACM Symp. on Principles of Database Systems*, 1987.

[77] C. Beeri and R. RaIllakrishnan. On the power of rnagic. In *ACM Syrnp. on Principles of Database Sy8terns, 1987.*

[78] D. Bell and J. GriIDson. *Distributed Database Systems*. AddisoIl-Wesley, 1992.

[79] J. Bentley and J. Friedman. Data structures for range searching. *ACM Cornputing Surveys*, 13(3):397–409, 1979.

[80] S. Berchtold, C. Bohm, and H.-P. Kriegel. The pyramid-tree: breaking the curse of dimensionality. In *ACM SIGMOD Conf. on the Management of Data, 1998.*

[81] P. Bernstein. Synthesizing third normal form relations from functional dependencies. *ACM Transactions on Database Systerns*, 1(4):277-298, 1976.

[82] P. Bernstein, B. Blaustein, and E. Clarke. Fast maintenance of sernantic integrity assertions using redundant aggregate data. In *Proc. Intl. Conf. on Very Large Databases*, 1980.

[83] P. Bernstein and D. Chiu. Using senli-joins to solve relational queries. *Journal of the ACM*, 28(1):25-40, 1981.

[84] P. Bernstein and N. Goodman. Tirnestamp-based algorithms for concurrency control in distributed database systems. In *Proc. Intl. Conf. on Very Large Databases, 1980.*

[85] P. Bernstein and N. Goodman. Concurrency control in distributed database systems. *ACM Computing Surveys*, 13(2):185-222, 1981.

[86] P. Bernstein and N. Goodlnan. Power of natural semijoins. *SIAM Journal of Computing*, 10(4):751–771, 1981.

[87] P. Bernstein and N. Goodulan. Multiversion concurrency control-Theory and algorithms. *ACM Transactions on Database Systerns*, 8(4):465-483, 1983.

[88] P. Bernstein, N. Goodnlan, E. Wong, C. Reeve, and J. Rothnie. Query processing in a systern for distributed databases (SDD-1). *ACM 1ransactions on Database Systerns*, 6(4):602–625, 1981.

[89] P. Bernstein, V. Hadzilacos, and N. Goodlnan. *Concurrency Cont'rol and Recovery in Database System,s*. Addison-vVesley, 1987.

[90] P. Bernstein and E. Newcomer. *Principles of Transaction Processing*. Morgan Kaufmann, 1997.

[91] P. Bernstein, D. Shiprnan, and J. Rothnie. Concurrency control in a SystCIll for distributed databases (SDD-l). *ACM Transactions on Database Systerns*, 5(1):18–51, 1980.

[92] P. Bernstein, D. Shiprnan, and \V. Wong. Forrnal aspects of serializability in database concurrency control. *IEEE Transactions on Software Engineering*, 5(3):203–216, 1979.

[93] K. Beyer, J. Goldstein, R. RaInakrishnan, and U. Shaft. When is nearest neighbor rneaningful? In *IEEE International Conference on Database Theory, 1999.*

[94] K. Beyer and R. Rarnakrishnan. BottOlIl-UP cornputatioll of sparse and iceberg cubes In *Proc. ACM SIGMOD Conf. on the Alanagernent of Data, 1999.*

[95] B. Bhargava, editor. *Concurrency Control and Reliability in Distributed Systems.* Van Nostrand Reinhold, 1987.

[96] A. Biliris. The performance of three database storage structures for r.nanaging large objects. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1992.*

[97] J. Biskup and B. Convent. A fonnal view integration method. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1986.*

[98] J. Biskup, U. Dayal, and P. Bernstein. Synthesizing independent database schenlas. In *Proc. ACM SIGMOD Conf. on the A'ianage7nent of Data, 1979.*

[99J D. Bitton and D. DevVitt. Duplicate record elimination in large data files. *ACM Transactions on Database System.s,* 8(2):255-265, 1983.

[100] J. Blakeley, P.-A. Larson, andF'. TOInpa. Efficiently updating lllaterialized views. In *Proc. ACM SIGN/OD Conf. on the Management of Data, 1986.*

[101] M. Blasgen and K. Eswaran. On the evaluation of queries in a database systenl. Technical report, IBM FJ (R.J1745), San Jose, 1975.

[102] P. Bohannon, D. Leinbaugh, R. Rastogi, S. Seshadri, A. Silberschatz, and S. Sudarshan. Logical and physical versioning in main memory databases. In *Proc. Intl. Conf. on Very Large Databases, 1997.*

(103] P. Bohannon, J. Freire, P. Roy, and J. Siuleon. From XML schema to relations: A cost-based approach to XML storage. In *Proceedings of ICDE, 2002.*

[104] P. Bonnet and D. E. Shasha. *Database Tuning: Pr'lnciples, Experirnents, and Troubleshooting Techniq1Les.* J\lIorgan Kaufrnann Publishers, 2002.

[105] G. Booch, 1. Jacobson, and J. Rurnbaugh. *The Unified Model'lng Language User Guide.* Addison-Wesley, 1998.

[106] A. Borodin, G. Roberts, J. Rosenthal, and P. Tsaparas. Finding authorities and hubs frorn link structures on Roberts G.O. the world wide web. In *World Wide Web Conference,* pages 415-429, 2001.

[107] R. Boyce and D. Chamberlin. SEQUEL: A structured English query language. In *PTOC. ACM SIGMOD Conf. on the Management of Data, 1974.*

[108] P. S. Bradley and U. M. Fayyad. Refining initial points for K-Means clustering. In *Pr'oc. Intl. Conlon Machine Learning,* pages 91-99. Morgan Kaufnlann, San :Francisco, CA, 1998.

[109] P. S. Bradley, U.N!. Fayyad, and C. Reina. Scaling clustering algorithnls to large databases. In *Pr·oc. Intl. Conlon Knowledge Discovery and Data Mining, 1998.*

[lID] K. Bratbergscngen. I-lashing rnethods and relational algebra operations. In *Pr·oc. Intl. ConI on Very Large Databases, 1984.*

[111] L. Brcilnan, J. H. Friechnan, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees.* Wadsworth, Belrnont. CA, 1984.

[112] Y. Breitbart, H. Garcia-l\/Iolina, and A. Silberschatz. Overvic\v of multidatabase transaction rnanagernent. In *Proc. Intl. C'onf. on Very Large Databases, 1992.*

[113] Y. Breitbart, A. Silberschatz, and G. Thornpson. Reliable transaction rllanagcrllent in a llmltidatabase systerll. In *Proc. ACM SIGMOD Conf. on the Management of Data,* 1990.

[114J Y. Breitbart, A. Silberschatz, and G. Thompson. An approach to recovery Inanagcrnent in a multidatabase system. In *Proc. IntI. Conf. on Very Large Databases,* 1992.

[115] S. Brin, R. Motwani, and C. Silverstein. Beyond Inarket baskets: Generalizing association rules to correlations. In *Proc. ACM SIGMOD Conf. on the Management of Data*, 1997.

[116] S. Brin and L. Page. The anatorny of a large-scale hypertextual web search engine. In *Proceedings of 7th World Wide Web Conference, 1998.*

(117) S. Brin, R. Motwani, J. D. lHlrnan, and S. Tsur. Dynaruic itertlset counting and inlplication rules for rnarket basket data. In *Proc. ACM SIGMOD lntl. Conf. on Management of Data*, pages 255···264. ACM Press, 1997.

[118] T. Brinkhoff, H.-P. Kriegel, and R. Schneider. Cornparison of approximations of cOlllplex objects used for approximation-ba..'3ed query processing in spatial database systerIls. In *Proc. IEEE IntZ. Conf. on Data Engineering, 1993.*

[119] K. Brown, M. Carey, and M. Livny. Goal-oriented buffer rnanagement revisited. In *Proc. ACM SIGMOD Conf. on the NJanagernent of Data, 1996.*

[120] N. Bruno, S. Chaudhuri, and L. Gravano. Top-k selection queries over relational databases: Mapping strategies and performance evaluation. *ACM Transactions on Database System,s,* To appear, 2002.

[121] F. Bry. Towards an efficient evaluation of general queries: Quantifier and disjunction processing revisited. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1989.*

[122] F. Bry and R. Manthey. Checking consistency of database constraints: A logical basis. In *Proc. Intl. Conf. on Very Large Databases, 1986.*

[123] P. Bunernan and E. Clemons. Efficiently rnonitoring relational databases. *ACM Transactions on Database Systerns,* 4(3), 1979.

[124] P. Bunernan, S. Davidson, G. Hillebrand, and D. Suciu. A query language and optimization techniques for unstructured data. In *Proc. ACM SIGMOD Conf. on Management of Data, 1996.*

[125] P. Buneman, S. Naqvi, V. Tannen, and L. Wong. Principles of prograrnrning with complex objects and collection types. *Theoretical Computer Science,* 149(1):3–48, 1995.

[126] D. Burdick, M. Calirnlim, and J. E. Gehrke. Mafia: A rnaxirnal frequent itemset algaritlull for transactional databases. In *Proc. Intl. Conf. on Data Engineering (JCDE).* IEEE Cornputer Society, 2001.

[127] M. Carey. Granularity hierarchies in concurrency control. In *ACM Symp. on Principles of Database Systern8,* 1983.

[128] M. Carey, D. Charnberlin, S. Narayanan, B. Vance, D. Doole, S. Rielau, R. Swagerrnan, and N. lVlattos. O-O, what's happening to DB2? In *Prnc. ACM SIGMOD Conf. on the Management of Data,* 1999.

[129] M. Carey, D. DeWitt, M. Franklin, N. Hall,N1. McAuliffe, .1. Naughton, D. Schuh, M. SOlOllIOIl, C. 'rau, O. Tsatalos, S. White, and M. Zwilling. Shoring up persistent applications. In *Proc. ACM SIGMOD Can]. on the Management of Data, 1994.*

[130] M. Carey, D. De\Vitt, G. Graefe, D. Haight, J. Richardson, D. Schuh, E. Shekita, and S. Vandenberg. The EXODUS Extensible DBMS project: An overview. In S. Zdonik and D. Maier, editors, *Readings in Object-Oriented Databases.* Morgan K.aufrnann, 1990.

[131] 1\'1. Carey, IJ. De\Vitt, and .1. Naughton. The 007 benchrnark. In *Proc. ACM SICA/OD Conj. on the Management of Data,* 1993.

[132] M. Carey, D. DeWitt, J. Naughton, M. Asgarian, J. Gehrke, andD. Shah. The BUGK\{ object-relational benchlnark. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1997.*

[133] M. Carey, D. DeWitt, J. Richardson, and E. Shekita. Object and file rnanageInent in the Exodus extensible database system. In *Proc. IntI. Conf. on Very Lar:qe Databases,* 1986.

[134] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaraul, E. Shekita, and S. Subramanian. XPERANTO: publishing object-relational data as XML. In *Pr'oceedings of the Third International Workshop on the Web and Databases,* May 2000.

[1:35] M. Carey and D. Kosslllan. On saying "Enough Already!" in SQL In *Proc. ACM SIGMOD Conf. on the Management of Data, 1997.*

[136] M. Carey and D. Kossrnan. Reducing the braking distance of an SQL query engine In *PTOC. Intl. Conf. on Very Large Databases, 1998.*

[137] M. Carey and M. LivllY. Conflict detection tradeoffs for replicated data. *ACM Transactions on Database Systerns,* 16(4), 1991.

[138] M. Casanova, L. Tuchennan, and A. F\utado. Enforcing inclusion dependencies and referential integrity. In *Proc. Intl. Conf. on Very Large Databases, 1988.*

[139] M. Casanova and M. Vidal. Towards a sound view integration lnethodology. In *ACM Symp. on Principles of Database Systems, 1983.*

[140] S. Castano, M. Fugini, G. Martella, and P. Samarati. *Database Security.* Addison-Wesley, 1995.

[141] R. Cattell. *The Object Database Standard: ODMG-93 (Release* 1.1). Morgan Kaufmann, 1994.

[142] S. Ceri, P. Fraternali, S. Paraboschi, and L. Tanca. Active rule lnanagement in Chimera. In J. Widom and S. Ceri, editors, *Active Database Systems.* Morgan Kaufmann, 1996.

[143] S. Ceri, G. Gottlob, and L. Tanca. *Logic Programming and Databases.* Springer Verlag, 1990.

[144] S. Ceri and G. Pelagatti. *Distributed Database Design: Principles and Systems.* McGraw-Hill, 1984.

[145] S. Ceri and J. Widom. Deriving production rules for constraint maintenance. In *Proc. Intl. Conf. on Very Large Databases, 1990.*

[146] F. Cesarini, M. Missikoff, and G. Soda. An expert systeln approach for database application tuning. *Data and Knowledge Engineering,* 8:35"'55, 1992.

[147] U. Chakravarthy. Architectures and rnonitoring techniques for active databases: An evaluation. *Data and Knowledge Engineering,* 16(1):1'-26, 1995.

[1.48] U. Chakravarthy, .1. Grant, and J. Minker. Logic-based approach to semantic query optimization. *ACM Tran8actions on Database Systern8,* 15(2):162····207, 1990.

[149] I). Charnberlill. *Using the New DB2.* Morgan Kaufrnann, 1996.

[150] D. Chaluberlin, M. Astrahan, M. Blasgen, J. Gray, W. King, B. Lindsay, R. Lode, .1. Mehl, T. Price, P. Selinger, M. Schkolnick, D. Slutz, I. Traiger, B. Wade, and R. Yost. A history and evaluation of System R *Comm7J:nication.s of the ACM,* 24(10):632--646, 1981.

[151] D. Chamberlin, M. Astrahan, K. Eswaran, P. Griffiths, R. Lorie, J. Mehl, P. Reisner, and B. Wade. Sequel 2: a unified approach to data definition, manipulation, and control. *IBM Journal of Research and Developrnent,* 20(6):560"""575, 1976.

[152] D. Charnberlin, D. Florescu, and .1. Robie. Quilt: an XML query language for heterogeneous data sources. In *Proceedings of WebDB,* Dallas, 'TX, May 2000.

[153] D. Chamberlin, D. Florescu, J. Robie, J. Simeon, and M. Stefanescu. XQuery: A query language for XML. World Wide Web ConsortiUJIl, `http://www.w3.`org!TR!xquery, Feb 2000.

[154] A. Chandra and D.Harel. Structure and complexity of relational queries. *J. Computer and System Sciences,* 25:99--128, 1982.

[155] A. Chandra and P. Ivlerlin. Optinlal ilnplernentation of conjunctive queries in relational databases. In *Proc. ACM SIGACT Syrnp. on Theory of Co'mp'uting, 1977.*

[156] M. Chandy, L. Haas, and J. Misra. Distributed deadlock detection. *ACM Transactions on Co'mputer Systems*, 1(3):144--156, 1983.

[157] C. Chang and D. Leu. Multi-key sorting as a file organization schenle when queries are not equally likely. In *Proc. Intl. Syrnp. on Database Systems for Advanced Applications,* 1989.

[158] D. Chang and D. Harkey. *Client/ server data access with Java and XML.* John Wiley and Sons, 1998.

[159] M. Charikar, S. Chaudhuri, R. Motwani, and V. R. Narasayya. Towards estimation error guarantees for distinct values. In *Proc. ACM Symposium on Principles of Database Systems,* pages 268–279. ACM, 2000.

[160] D. Chatziantoniou and K. Ross. Groupwise processing of relational queries. In *Proc. Inti. Conf. on Very Large Databases, 1997.*

[161] S. Chaudhuri and U. Dayal. An overview of data warehousing and OLAP technology. *SIGMOD Record*, 26(1):65–74, 1997.

[162] S. Chaudhuri and D. Madigan, editors. *Proc. ACM SIGKDD Inti. ConfeTence on Knowledge Discovery and Data lvIining.* ACIvI Press, 1999.

[163] S. Chaudhuri and V. Narasayya. An efficient cost-driven index selection tool for Microsoft SQL Server. In *Proc. Inti. Conf. on Very Large Databases, 1997.*

[164] S. Chaudhuri and V. R. Narasayya. Autoadrnin 'what-if' index analysis utility. In *Proc. ACM SIGMOD Inti. Conf. on lvIanagernent of Data, 1998.*

[165] S. Chaudhuri and K. Shirrl. Optimization of queries with user-defined predicates. In *Pr'Oc. Intl. Conf. on Very Large Databases, 1996.*

[166] S. Chaudhuri and K. Shirn. Optimization queries with aggregate views. In *Intl. Conf. on Extending Database Technology, 1996.*

[167] S. Chaudhuri, G. Das, and V. R. Narasayya. A robust, optirrlization-hased approach for approximate answering of aggregate queries. In *Proc. ACM SIG1v/OD Conf. on the Management of Data, 2001.*

[168] J. Cheiney, P. Faudenlay, R. Michel, and J. Thevenin. A reliable parallel backend using rnultiattribute clustering and select-join operator. In *Proc. Intl. Conf. on Very Large Databases, 1986.*

[169] C. Chen and N.Roussopoulos. Adaptive database buffer rnanagernent using query feedback. In *Proc. Inti. Conj. on Very Lar:qe Databases,* 1993.

[170J C. Chen and N. Roussopoulos. Adaptive selectivity estimation using query feedback. In *Proc. ACM SIGMOD Conf. on the Manage1nent of Data, 1994.*

[171] P. M. Chen, E. K. Lee, G. A. Gibson, R. H. Katz, andD. A. Patterson. RAID: High-perforrnance, reliable secondary storage. *AClvI Computing Surveys*, 26(2):14,5·..185, June 1994.

[172} P. P. Chen. The entity-relationship rnodel-········towarda unified view of data. *ACM Transactions on Database System,s,* 1(1):9--36, 1976.

[173] Y. Chen, G. Dong, J. Han, B. W. Wah, and J.vVang. Multi-dimensional regression analysis of tinIe-series data strearllS. In *Proc. Intl. Conf. on Very Large Data Bases*, 2002.

[174] D. W. Cheung, .1. Han, V. T. Ng, and C. Y. Wong. Maintenance of discovered association rules in large databases: An incrernental updating technique. In *Proc. Int. Conf. Data Engineering*, 1996.

[175] D. W. Cheung, V. T. Ng, and B. W. Tarn Maintenance of discovered knowledge: A case in rnlllti-level association rules. In *Proc. Intl. Conf. on Knowledge Discovery and Data Mining.* AAAI Press, 1996.

[176] D. Childs. Feasibility of a set theoretical data structure—A general structure based on a reconstructed definition of relation. *Proc. Tri-annual IFIP Conference,* 1968.

[177] D. Chimenti, R. Garnboa, R. Krishnarnurthy, S. Naqvi, S. Tsur, and C. Zaniolo. The lell system prototype. *IEEE Tra'nsactions on Knowledge and Data Engineering,* 2(1):76–90, 1990.

[178] F. Chin and G. Ozsoyoglu. Statistical database design. *ACM TTansactions on Database Systerns,* 6(1):113--139, 1981.

[179] T.-C. Chiueh and L. Huang. Efficient real-time index updates in text retrieval systems.

[180] J. ChOillicki. Real-time integrity constraints. In *ACM Symp. on Principles of Database Syste'ms, 1992.*

[181] H.-T. Chou and D. DeWitt. An evaluation of buffer rnanagelnent strategies for relational database systerns. In *Proc. Intl. Conf. on Very Large Databases, 1985.*

[182] P. Chrysanthis and K. Ramarnritharn. Acta: A framework for specifying and reasoning about transaction structure and behavior. In *Proc. ACM SIGN/OD Conf. on the lvlanagement of Data, 1990.*

[183] F. Chu, J. Halpern, and P. Seshadri. Least expected cost query optinlization: An exercise in utility *ACM Symp. on Principles of Database System,s, 1999.*

[184] F. Civelek, A. Dogac, and S. Spaccapietra. An expert systern approach to view definition and integration. In *Proc. Entity-Relationship ConfeTence,* 1988.

[185] R.. Cochrane, H. Pirahesh, and N. Mattos. Integrating triggers and declarative constraints in SQL database systems. In *Pr'Oc. Intl. Conf. on Very Large Databases, 1996.*

[186] CODASYL. *Report of the CODASYL Data Base Task Group.* ACM, 1971.

[187]E. Codd. A relational lllodel of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[188] E. Codd. Further norrnalization of the data base relational rnodeL In R. Rustin, editor, *Data Base Systems.* Prentice Hall, 1972.

[189J E. Codd. Relational cOlnpleteness of data base sub-languages. Inll. Rustin, editor, *Data Base Systems.* Prentice Hall, 1972.

[190] E. Codd. Extending the database relational lnodel to capture rnore IllCC1ning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[191] E. Codd. Twelve rules for on-line analytic processing. *Computerworld*, April 13 1995.

[192] L. Colby, T. Griffin, L. Libkin, 1. Mumick, anei H. 'Trickey. Algorith.IllS for deferred view rnaintenance. In *Prnc. ACM SIGMOD ConI on the Management of Data*, 1996.

[193] L. Colby, A. Kawaguchi, D. Lieuwen, 1. M111nick, and K. Ross. Supporting multiple view rnaintenance policies: Concepts, algorithnls, and performance analysis. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1997.*

[194} D. COIner. The ubiquitous B-tree. *ACM C. Surveys*, 11(2):121··1:37, 1979.

[195] D. Connolly, editor. *XML Principles, Tools and Techniques.* O'Reilly & Associates, Sebastopol, USA, 1997.

[196J B. Cooper, N. Sample, M. J. Franklin, G. R. Hjaltason, and M. Shadmon. A fast index for senlistructured data. In *Proceedings of VLDB, 2001.*

[197J D. Copeland and D. Maier. Making SMALLTALK a database systerll. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1984.*

[198] G. Cornell and K. Abdali. *CGI Prograrnm'ing With Java.* PrenticeHall, 1998.

[199} C. Cortes, K. Fisher, D. Pregibon, and A. Rogers. Hancock: a language for extracting signatures fronl data strearllS. In *Proc. ACM SIGKDD Inti. Conference on Knowledge Discovery and Data Mining*, pages 9–17. AAAI Press, 2000.

[200] J. Daenlen and V. RijrrleIl. *The Design of Rijndael: AES -The Advanced Encryption Standard (Information Security and Cryptography).* Springer Verlag, 2002.

[201] M. Datal', A. Gionis, P. Indyk, and R.. Motwani. Maintaining stream statistics over sliding windows. In *PTOC. of the Annual ACM-SIAM Symp. on Discrete Algorithms,* 2002.

[202] C. Date. A critique of the SQL database language. *ACM SIGM·OD Record, 14(3):8-54,* 1984.

[203] C. Date. *Relational Database: Selected Writings.* Addison-Wesley, 1986.

[204] C. Date. *An Introduction to Database Systems.* Addison-Wesley, 7 edition, 1999.

[205] C. Date and R. Fagin. SiIllpie conditiolls for guaranteeing higher norrnal forms In relational databases. *ACM Transactions on Database Systerns,* 17(3), 1992.

[206] C. Date and D. McGoveran. *A Guide to Sybase and SQL Server.* Addison-Wesley, 1993.

[207] U. Dayal and P. Bernstein. On the updatability of relational views. In *Proc. Intl. Conf. on Very Large Databases, 1978.*

[208] U. Dayal and P. Bernstein. On the correct translation of update operations on relational views. *ACM Transactions on Database Systems,* 7(3), 1982.

[209] P. DeBra and J. Paredaens. Horizontal decompositions for handling exceptions to FDs. In H. Gallaire, .1. Minkel', and J.-M. Nicolas, editors, *Advances in Database Theory,.* PlenurIl Press, 1981.

[210] J. Deep and P. Holfelder. *Developing CGI applications with Perl.* Wiley, 1996.

[211] C. Delobel. Norrrialization and hierarchial dependencies in the relational data model. *ACM TTansactions on Database Systerns,* 3(3):201–222, 1978.

[212] D. Denning. Secure statistical databases with randOlIl sC1nlple queries. *ACM Transactions on Database Systems,* 5(3):291–315, 1980.

[213] D. E. Denning. *Cryptography and Data Security.* AddisOI1-Wesley, 1982.

[214] M. Derr, S. Nlorishita, and G. Phipps. The glue-nail deductive database systern: Design, implernentation, and evaluation. *VLDB Journal*, 3(2):123-–160, 1994.

[215] A. Deshpailde. An iruplelnentation for nested relational databases. Technical report, PhD thesis, Indiana University, 1989.

[216] P. I)eshpande, K. Ramasamy, A. Shukla, and J. F. Naughton. Caching rIlultidirnensional queries using chunks. In *Proc. ACM SIGMOD Intl. Conf. on Management of Data, 1998.*

[217] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Sueiu. XML-QL: A query language for XML. WorldWide Web Consortium, `http://www.w3`.org/TR/NOTE-xml-ql, Aug 1998.

[218] O. e. a. Deux. The story of 02. *IEEE Transactions on Knowledge and Data Engineering,* 2(1), 1990.

[219] D. DeWitt, II.-T. Chou, R. Katz, and A. Klug. Design and inlplenlcntation of the Wisconsin Storage Systern. *Software Practice a'nd Experience,* 15(10):943--962, 1985.

[220] D. DeWitt, R. Gerber, G. Graefe, M. Heytens, K. Kumar, and M. Muralikrishna. Ganuua-····-··Ahigh perforrnance dataflow database Inachine. In *Proc. Intl. Conf. on Very Large Databases, 1.986.*

[221] D. DeWitt and J. Gray. Parallel database systenls: The future of high-perfornlance database systerIls. *Cornm:unications of the ACM,* 35(6):85-98, 1992.

[222] D. DeWitt, R. Katz, F. Olken, L. Shapiro, M. Stonebraker, and D. Wood. Inlplelnentation techniques for rnain menlory databases. In *Proc. ACM SIGMOD Conf. on the Management of Dat.a,* 1984.

(223] D. DeWitt, J. Naughton, and D. Schneider. Parallel sorting on a shared-nothing architecture using probabilistic splitting. In *Proc. Conf. on Parallel and Distributed Inforrnation Systerns, 1991.*

[224] D. DeWitt, J. Naughton, D. Schneider, and S. Seshadri. Practical skew handling in parallel joins. In *Proc. Inti. Conf. on Very Large Databases, 1992.*

[225] O. Diaz, N. Paton, and P. Gray. Rule rnanagenlent in object-oriented databases: A uniform approach. In *Proc. Ina. Conf. on Very Large Databases, 1991.*

[226] S. Dietrich. Extension tables: Merno relations in logic programming. In *Proc. Intl. Symp. on Logic Programming, 1987.*

[227] W. Diffie and M. E. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory,* 22(6):644-654, 1976.

[228] P. Domingos and G. Hulten. Mining high-speed data strearns. In *Proc. ACM 8IGI(DD Inti. ConfeTence on }(nowledge Discovery and Data Mining.* AAAI Press, 2000.

[229] D. Donjerkovic and R. Ramakrishnan. Probabilistic optiInization of top N queries In *PTOC. Inti. Conf. on Very Large Databases, 1999.*

[230] W. Du and A. Eltnagarrnid. Quasi-serializability: A correctness criterion for global concurrency control in interbase. In *Proc. Intl. Conf. on Very Large Databases, 1989.*

(231] W. Du, R. Krishnarnurthy, and M.-C. Shan. Query optiruization in a heterogeneous DBIVIS. In *PTOC. Intl. ConI on VeTy LaTge Database8, 1992.*

[232] R. C. Dubes and A. .Jain. *Clustering f\1ethodologies in Exploratory Data Analysis, Advances in Computers.* Acadelnic Press, New York, 1980.

[233] N. Duppe!. Parallel SQL on TANDEM 's NonStop SQL. *IEEE COMPCON, 1989.*

[2:34] H. Edelstein. The challenge of replication, Parts 1 and 2. *DBMS: Database and Client-Server Solutions, 1995.*

[235J \V.Effelsberg and T. Haerder. Principles of database buffer rnanagement. *ACM Transactions on Database Systems,* 9(4):560-595, 1984.

[236] M. H. Eich. A classification and cOITlparison of rnain lTlCrnory database recovery techniques. In *Proc. IEEE IntZ. Conf. on Data Engineering,* 1987.

[237] A. Eisenberg and J. Melton. SQL:1999 , forrnerly kno\vn as SQL:3 *ACM SIGMOD Record,* 28(1):131-138, 1999.

[238] A. El Abbadi. Adaptive protocols for rnanaging replicated distributed databases. In *IEBB Symp. on Parallel and Distributed Processing, 1991.*

[239] A. EI Abbadi, D. Skeen, and F. Cristiano An efficient, fault-tolerant protocol for replicated data managenlent. In *ACM Symp. on Principles of Database Systems, 1985.*

[240] C. Ellis. Concurrency in Linear Hashing. *ACM Transactions on Database Systems*, 12(2):195·····217,1987.

[241] A. Ehnagarrnid. *Database Transaction Models for Advanced Applications.* Morgan Kauflllann, 1992.

[242] A. Elrnagannid, J. ,Hng, W. Kinl, O. Bukhres, and A. Zhang. Global cOllunitability in liluitidatabase systems. *IEEE Transactions on Knowledge and Data Engineering*, 8(5):816824, 1996.

[243] A. Elrnagarrnid, A. Sheth, and M. Liu. Deadlock detection algorithms in distributed database systenls. In *Proc. .IEEE Intl. Conf. on Data Engineering, 1986.*

[244] R. Elmasri and S. Navathe. Object integration in database design. In *Proc. IEEE Intl. Conf. on Data Engineering, 1984.*

[245] R. Ehnasri and S. Navathe. *Fundamentals of Database Systems.* Benjamin-Curnrnings, 3 edition, 2000.

[246] R. Epstein. Techniques for processing of aggregates in relational database systeIlls. Technical report, DC-Berkeley, Electronics Research Laboratory, M798, 1979.

[247] R. Epstein, M. Stonebraker, and E. Wong. Distributed query processing in a relational data base system. In *Proc. AC!vI SIGMOD Conf. on the !vlanagement of Data, 1978.*

[248] M. Ester, H.-P. Kriegel, J. Sander, M. Wimmer, and X. Xu. Incremental clustering for mining in a data warehousing environment. In *Proc. Intl. Conf. On Very Large Data Bases, 1998.*

[249] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. Intl. ConI. on Knowledge Discovery in Databases and Data Mining, 1995.*

[250] J\1. Ester, H.-P. Kriegel, and X. Xu. A database interface for clustering in large spatial databases. In *Proc. Intl. ConI. on Knowledge Discovery in Databases and Data Mining,* 1995.

[251] K. Eswaran and D. Chamberlin. Functional specification of a subsysteIll for data base integrity. In *Proc. Intl. ConI. on Very Lar:qe Databases, 1975.*

[252] K. Eswaran, J. Gray, R. Lorie, and 1. Traiger. The notions of consistency and predicate locks in a data base systern. *Communications of the AC!v[*, 19(11):624–633, 1976.

[253] R.Fagin. Multivalued dependencies and a new nornml fonn for relational databases. *ACM Transactions on Database Systerns*, 2(3):262–278, 1977.

[254] R. Fagin. Normal fonns and relational database operators. In *Proc. ACM SIGMOD Conf. on the Management of Data, 1979.*

R. Fagin. A nonnal form for relational databases that is based on dornains and keys. *ACM Transactions on Database Systerns*, 6(3):387–415, 198!.

[256] R. Fagin, J. Nievergelt, N. Pippenger, and H. Strong. Extendible Hashing--·a fast access rnethod for dynarnic files. *ACM Transactions on Database Systems*, 4(3), 1979.

[257] C. Faloutsos. Access rnethods for text. *ACM Computing Surveys*, 17(1):49–74, 1985.

[258] C. Faloutsos. *Searching Multimedia Databases by Content* Kluwer Acadernic, 1996.

[259] C. Faloutsos and S. Christodoulakis. Signature files: An access rnethod for docurnents and its analytical perforrnance evaluation. *ACM Transactions on Oifice Inforrnation Systems*, 2(4):267288, 1984.

[260] C. Faloutsos and H. Jagadish. On B-Tree indices for skewed distributions. In *Proc. Inti. Conf. on Very Large Databases, 1992.*

[261] C. Faloutsos, R. Ng,and T. Sellis. Predictive load control for flexible buffer allocation. In *Proc. Intl. Conf. on Very Large Databases, 1991.*

[262] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence lnatching in titne-series databases. In *Proc. ACM SIGMOD Conf. on the Management of Data,* 1994.

[263] C. Faloutsos and S. Rasenlan. Fractals for secondary key retrieval. In *ACM Symp. on Principles of Database 8yste'ms, 1989.*

[264] M. Fang, N. ShivakuIIlar, H. Garcia-Molina, R. Motwani, and J. D. Ullrnan. Cornputing iceberg queries efficiently. In *Proc. Intl. Conf. On Very Large Data Bases, 1998.*

[265] U. Fayyad, G. Piatetsky-Shapiro, and P. Srnyth. The KDD process for extracting useful knowledge from volumes of data. *Co'rnmunications of the ACM*, 39(11):27--34, 1996.

[266] U. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, editors. *Advances in Knowledge Discovery and Data Mining.* MIT Press, 1996.

[267] U. Fayyad and E. Simoudis. Data mining and knowledge discovery: T'utorial notes. In *Inti. Joint Conf. on Artificial Intelligence, 1997.*

[268] U. M. Fayyad and R. Uthurusamy, editors. *Proc. Intl. ConI 'on Knowledge Discovery and Data Mining.* AAAI Press, 1995.

[269] M. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. STRUDEL: A Web site management system. In *Proc. ACM SIGMOD Conf. on Management of Data, 1997.*

[270] M. Fernandez, D. Florescu, A. Y. Levy, and D. Suciu. A query language for a Web -site management system. *SIGMOD Record (ACM Special Interest Group on Management of Data),* 26(3):4-11, 1997.

[271] M. Fernandez, D. Suciu, and W. Tan. SilkRoute: trading between relations and XML. In *Proceedings of the WWW9, 2000.*

[272] S. Finkelstein, M. Schkolnick, and P. Tiberio. Physical database design for relational databases. *IBM Research Review RJ5034*, 1986.

[273] D. Fishrnan, D. Beech, H. Cate, E. Chow, T. Connors, J. Davis, N. Derrett, C. Hoch, W. Kent, P. Lyngbaek, B. Nlahbod, M..-A. Neirnat, T. Ryan, and M.-C. Shan. Iris: an object-oriented database rnanagernent system *ACM Transactions on Office Infor"mation Systerns,* 5(1):48--69, 1987.

[274J C. Flerning and B. von Halle. *IIandbook of Relational Database Desig'n.* Addison-Wesley, 1989.

[275] D. Florescu, A. Y. Levy, and A. O. IVlendelzon. Database techniques for the World-Wide Web: A survey. *SIGlvl0D Record (ACM Special Interest Group on Management of Data),* 27(3):59-74, 1998.

[276] W. Ford and M. S. Baum. *Secure Electronic Commerce: Building the Infrastructure for Digital Signatures and Bnc'ryption (2nd Edition).* Prentice Hall, 2000.

[277] F. F'otouhi and S. Prarnanik. Optimal secondary storage access sequence for perfonning relational join. *IEEE Transactions on Knowledge and Data Engineering,* 1(3):318-328, 1989.

[278J :LVl. Fowler and K. Scott. *UML D'lstilled: Applying the Standard Object Modeling Language.* Addison-\Nesley, 1999.

[279] W. B. Frakes and R. Baeza-Yates, editors. *Inforrnation Retrieval: Data Structures and Algorithms.* PrenticeHall, 1992.

[280J P. Franaszek, J. Robinson, and A. Thornasian. Concurrency control for high contention environrnents. *ACM Transactions* on *Database Systems,* 17(2), 1992.

[281] P. Franazsek, J. Robinson, and A. Thornasian. Access invariance and its use in high contention enviroIlrnents. In *Proc. IEEE International Conference on Data Eng'ineering,* 1990.

[282] M. Franklin. Concurrency control and recovery. In *Handbook of Computer Science, A.B. Tucker (ed.)) eRe Press, 1996.*

[283] M. :FrankliIl, M. Carey, and M. Livny. Local disk caching for client-server database systerlls. In *Proc. Intl. Conj. on Very Large Databases, 1993.*

[284] M. Franklin, B. Jonsson, and D. Kosslnan. Perfonnance tradeoffs for client-server query processing. In *Proc. ACM SIGMOD Conj. on the Managernent of Data, 1996.*

[285] P. Fraternali and L. Tanca. A structured approach for the definition of the semantics of active databases. *AeM Transactions on Database Systems,* 20(4):414---471, 1995.

[286] M. W. Freeston. The BANG file: A new kind of Grid File. In *Proc. ACM SIGMOD Conj. on the JvIanagement of Data, 1987.*

[287] .1. Freytag. A rule-based view of query optimization. In *Proc. ACJvI SIGJvIOD Conj. on the Managernent of Data, 1987.*

[288] O. Friesen, A. Lefebvre, and L. Vieille. VALIDITY: Applications of a DOOD system. In *IntZ. Can/. on Extending Database Technology, 1996.*

[289] J. Fry and E. Sibley. Evolution of data-base management systems. *ACM Computing Surveys,* 8(1):7-42, 1976.

[290] N. Fuhr. A decision-theoretic approach to database selection in networked ir. *AClvI Transactions on Database Systems,* 17(3), 1999.

[291] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. Mining optinlized association rules for numeric attributes. In *ACJvI Syrnp. on .Principles of Database Systems, 1996.*

[292] A. F\utado and M. Casanova. Updating relational views. In *Query Processing in Database Systems.* eds. W. Kiln, D.S. Reiner and D.S. Batory, Springer-Verlag, 1985.

[293] S. Fushin1i, M. Kitsuregawa, and H. Tanaka. An overview of the systems software of a parallel relational database machine: Grace. In *Proc. Intl. Conf. on Very Large Databases, 1986.*

[294] V. Gaede and O. Guenther. Multidinlensional access rnethods. *C01nputing Surveys,* 30(2):170–231, 1998.

[295] H. Gallaire, J. Minker, and J.-M. Nicolas (eds.). *Advances in Database Theory, Vols. 1 and* 2. Plenum Press, 1984.

[296] H. Gallaire and J. Minker (cds.). *Logic and Data Bases.* Plcnurn Press, 1978.

[297] S. Ganguly, W. Hasan, and R. Krishnarnurthy. Query optirnizatioll for parallel execution. In *PTOC. ACM SIGMOD Conj. on the Management of Data, 1992.*

[298J R. Ganski and H. Wong. Optirnization of nested SQL queries revisited. In *PTOC. ACM SIGMOD Conf. on the Management of Data, 1987.*

[299) V. Ganti, .1. Gehrke, and R. Rmnakrishnan. DenlOu: rnining and rnonitoring evolving data. *IEEE Transactions on Knowledge and Data Engineering,* 13(1), 2001.

[:300] V. Ganti, J. Gehrke, R. Ramakrishnan, and W.-Y. Loh. Focus: a framework for rneasuring changes in data characteristics. In *Proc. ACM Symposium on Principles of Database Systems,* 1999.