

part **11**

**Advanced Database Models,  
Systems, and Applications**

This page intentionally left blank

## Enhanced Data Models: Introduction to Active, Temporal, Spatial, Multimedia, and Deductive Databases

As the use of database systems has grown, users have demanded additional functionality from these software packages; increased functionality would make it easier to implement more advanced and complex user applications. Object-oriented databases and object-relational systems do provide features that allow users to extend their systems by specifying additional abstract data types for each application. However, it is useful to identify certain common features for some of these advanced applications and to create models that can represent them. Additionally, specialized storage structures and indexing methods can be implemented to improve the performance of these common features. Then the features can be implemented as abstract data types or class libraries and purchased separately from the basic DBMS software package. The term **data blade** has been used in Informix and **cartridge** in Oracle to refer to such optional submodules that can be included in a DBMS package. Users can utilize these features directly if they are suitable for their applications, without having to reinvent, reimplement, and reprogram such common features.

This chapter introduces database concepts for some of the common features that are needed by advanced applications and are being used widely. We will cover *active rules* that are used in active database applications, *temporal concepts* that are used in temporal database applications, and, briefly, some of the issues involving *spatial databases* and *multimedia databases*. We will also discuss *deductive databases*. It is important to note that each of these topics is very broad, and we give

only a brief introduction to each. In fact, each of these areas can serve as the sole topic of a complete book.

In Section 26.1, we introduce the topic of active databases, which provide additional functionality for specifying **active rules**. These rules can be automatically triggered by events that occur, such as database updates or certain times being reached, and can initiate certain actions that have been specified in the rule declaration to occur if certain conditions are met. Many commercial packages include some of the functionality provided by active databases in the form of **triggers**. Triggers are now part of the SQL-99 and later standards.

In Section 26.2, we introduce the concepts of **temporal databases**, which permit the database system to store a history of changes and allow users to query both current and past states of the database. Some temporal database models also allow users to store future expected information, such as planned schedules. It is important to note that many database applications are temporal, but they are often implemented without having much temporal support from the DBMS package—that is, the temporal concepts are implemented in the application programs that access the database. The ability to create and query temporal data has been added to the SQL standard in SQL:2011 and is available in the DB2 system, but we do not discuss it here. The interested reader is referred to the end-of-chapter bibliography.

Section 26.3 gives a brief overview of **spatial database** concepts. We discuss types of spatial data, different kinds of spatial analyses, operations on spatial data, types of spatial queries, spatial data indexing, spatial data mining, and applications of spatial databases. Most commercial and open source relational systems provide spatial support in their data types and query languages as well as providing indexing and efficient query processing for common spatial operations.

Section 26.4 is devoted to multimedia database concepts. **Multimedia databases** provide features that allow users to store and query different types of multimedia information, which includes **images** (such as pictures and drawings), **video clips** (such as movies, newsreels, and home videos), **audio clips** (such as songs, phone messages, and speeches), and **documents** (such as books and articles). We discuss automatic analysis of images, object recognition in images, and semantic tagging of images.

In Section 26.5, we discuss deductive databases,<sup>1</sup> an area that is at the intersection of databases, logic, and artificial intelligence or knowledge bases. A **deductive database system** includes capabilities to define (**deductive**) **rules**, which can deduce or infer additional information from the facts that are stored in a database. Because part of the theoretical foundation for some deductive database systems is mathematical logic, such rules are often referred to as **logic databases**. Other types of systems, referred to as **expert database systems** or **knowledge-based systems**, also incorporate reasoning and inferencing capabilities; such systems use techniques

---

<sup>1</sup>Section 26.5 is a summary of Deductive Databases. The full chapter from the third edition, which provides a more comprehensive introduction, is available on the book's Web site.

that were developed in the field of artificial intelligence, including semantic networks, frames, production systems, or rules for capturing domain-specific knowledge. Section 26.6 summarizes the chapter.

Readers may choose to peruse the particular topics they are interested in, as the sections in this chapter are practically independent of one another.

## 26.1 Active Database Concepts and Triggers

Rules that specify actions that are automatically triggered by certain events have been considered important enhancements to database systems for quite some time. In fact, the concept of **triggers**—a technique for specifying certain types of active rules—has existed in early versions of the SQL specification for relational databases, and triggers are now part of the SQL-99 and later standards. Commercial relational DBMSs—such as Oracle, DB2, and Microsoft SQLServer—have various versions of triggers available. However, much research into what a general model for active databases should look like has been done since the early models of triggers were proposed. In Section 26.1.1, we will present the general concepts that have been proposed for specifying rules for active databases. We will use the syntax of the Oracle commercial relational DBMS to illustrate these concepts with specific examples, since Oracle triggers are close to the way rules are specified in the SQL standard. Section 26.1.2 will discuss some general design and implementation issues for active databases. We give examples of how active databases are implemented in the STARBURST experimental DBMS in Section 26.1.3, since STARBURST provides for many of the concepts of generalized active databases within its framework. Section 26.1.4 discusses possible applications of active databases. Finally, Section 26.1.5 describes how triggers are declared in the SQL-99 standard.

### 26.1.1 Generalized Model for Active Databases and Oracle Triggers

The model that has been used to specify active database rules is referred to as the **event-condition-action (ECA)** model. A rule in the ECA model has three components:

1. The **event(s)** that triggers the rule: These events are usually database update operations that are explicitly applied to the database. However, in the general model, they could also be temporal events<sup>2</sup> or other kinds of external events.
2. The **condition** that determines whether the rule action should be executed: Once the triggering event has occurred, an *optional* condition may be evaluated. If *no condition* is specified, the action will be executed once the event

---

<sup>2</sup>An example would be a temporal event specified as a periodic time, such as: Trigger this rule every day at 5:30 a.m.

**EMPLOYEE**

Name	<u>Ssn</u>	Salary	Dno	Supervisor_ssn
------	------------	--------	-----	----------------

**Figure 26.1**

A simplified COMPANY database used for active rule examples.

**DEPARTMENT**

Dname	<u>Dno</u>	Total_sal	Manager_ssn
-------	------------	-----------	-------------

occurs. If a condition is specified, it is first evaluated, and only *if it evaluates to true* will the rule action be executed.

3. The **action** to be taken: The action is usually a sequence of SQL statements, but it could also be a database transaction or an external program that will be automatically executed.

Let us consider some examples to illustrate these concepts. The examples are based on a much simplified variation of the COMPANY database application from Figure 5.5 and are shown in Figure 26.1, with each employee having a name (Name), Social Security number (Ssn), salary (Salary), department to which she is currently assigned (Dno, a foreign key to DEPARTMENT), and a direct supervisor (Supervisor\_ssn, a (recursive) foreign key to EMPLOYEE). For this example, we assume that NULL is allowed for Dno, indicating that an employee may be temporarily unassigned to any department. Each department has a name (Dname), number (Dno), the total salary of all employees assigned to the department (Total\_sal), and a manager (Manager\_ssn, which is a foreign key to EMPLOYEE).

Notice that the Total\_sal attribute is really a derived attribute whose value should be the sum of the salaries of all employees who are assigned to the particular department. Maintaining the correct value of such a derived attribute can be done via an active rule. First we have to determine the **events** that *may cause* a change in the value of Total\_sal, which are as follows:

1. Inserting (one or more) new employee tuples
2. Changing the salary of (one or more) existing employees
3. Changing the assignment of existing employees from one department to another
4. Deleting (one or more) employee tuples

In the case of event 1, we only need to recompute Total\_sal if the new employee is immediately assigned to a department—that is, if the value of the Dno attribute for the new employee tuple is not NULL (assuming NULL is allowed for Dno). Hence, this would be the **condition** to be checked. A similar condition could be checked for event 2 (and 4) to determine whether the employee whose salary is changed (or who is being deleted) is currently assigned to a department. For event 3, we will always execute an action to maintain the value of Total\_sal correctly, so no condition is needed (the action is always executed).

The **action** for events 1, 2, and 4 is to automatically update the value of `Total_sal` for the employee's department to reflect the newly inserted, updated, or deleted employee's salary. In the case of event 3, a twofold action is needed: one to update the `Total_sal` of the employee's old department and the other to update the `Total_sal` of the employee's new department.

The four active rules (or triggers) R1, R2, R3, and R4—corresponding to the above situation—can be specified in the notation of the Oracle DBMS as shown in Figure 26.2(a). Let us consider rule R1 to illustrate the syntax of creating triggers in Oracle. The `CREATE TRIGGER` statement specifies a trigger (or active rule) name—`Total_sal1` for R1. The `AFTER` clause specifies that the rule will be triggered *after* the events that trigger the rule occur. The triggering events—an insert of a new employee in this example—are specified following the `AFTER` keyword.<sup>3</sup>

The `ON` clause specifies the relation on which the rule is specified—`EMPLOYEE` for R1. The *optional* keywords `FOR EACH ROW` specify that the rule will be triggered *once for each row* that is affected by the triggering event.<sup>4</sup>

The *optional* `WHEN` clause is used to specify any conditions that need to be checked after the rule is triggered, but before the action is executed. Finally, the action(s) to be taken is (are) specified as a PL/SQL block, which typically contains one or more SQL statements or calls to execute external procedures.

The four triggers (active rules) R1, R2, R3, and R4 illustrate a number of features of active rules. First, the basic **events** that can be specified for triggering the rules are the standard SQL update commands: `INSERT`, `DELETE`, and `UPDATE`. They are specified by the keywords **`INSERT`**, **`DELETE`**, and **`UPDATE`** in Oracle notation. In the case of `UPDATE`, one may specify the attributes to be updated—for example, by writing **`UPDATE OF Salary, Dno`**. Second, the rule designer needs to have a way to refer to the tuples that have been inserted, deleted, or modified by the triggering event. The keywords **`NEW`** and **`OLD`** are used in Oracle notation; `NEW` is used to refer to a newly inserted or newly updated tuple, whereas `OLD` is used to refer to a deleted tuple or to a tuple before it was updated.

Thus, rule R1 is triggered after an `INSERT` operation is applied to the `EMPLOYEE` relation. In R1, the condition (**`NEW.Dno IS NOT NULL`**) is checked, and if it evaluates to true, meaning that the newly inserted employee tuple is related to a department, then the action is executed. The action updates the `DEPARTMENT` tuple(s) related to the newly inserted employee by adding their salary (**`NEW.Salary`**) to the `Total_sal` attribute of their related department.

Rule R2 is similar to R1, but it is triggered by an `UPDATE` operation that updates the `SALARY` of an employee rather than by an `INSERT`. Rule R3 is triggered by an update to the `Dno` attribute of `EMPLOYEE`, which signifies changing an employee's assignment from one department to another. There is no condition to check in R3, so the

---

<sup>3</sup>As we will see, it is also possible to specify `BEFORE` instead of `AFTER`, which indicates that the rule is triggered *before the triggering event is executed*.

<sup>4</sup>Again, we will see that an alternative is to trigger the rule *only once* even if multiple rows (tuples) are affected by the triggering event.

**Figure 26.2**

Specifying active rules as triggers in Oracle notation. (a) Triggers for automatically maintaining the consistency of Total\_sal of DEPARTMENT.

(b) Trigger for comparing an employee's salary with that of his or her supervisor.

```
(a) R1: CREATE TRIGGER Total_sal1
        AFTER INSERT ON EMPLOYEE
        FOR EACH ROW
        WHEN ( NEW.Dno IS NOT NULL )
            UPDATE DEPARTMENT
            SET Total_sal = Total_sal + NEW.Salary
            WHERE Dno = NEW.Dno;

R2: CREATE TRIGGER Total_sal2
        AFTER UPDATE OF Salary ON EMPLOYEE
        FOR EACH ROW
        WHEN ( NEW.Dno IS NOT NULL )
            UPDATE DEPARTMENT
            SET Total_sal = Total_sal + NEW.Salary - OLD.Salary
            WHERE Dno = NEW.Dno;

R3: CREATE TRIGGER Total_sal3
        AFTER UPDATE OF Dno ON EMPLOYEE
        FOR EACH ROW
        BEGIN
            UPDATE DEPARTMENT
            SET Total_sal = Total_sal + NEW.Salary
            WHERE Dno = NEW.Dno;
            UPDATE DEPARTMENT
            SET Total_sal = Total_sal - OLD.Salary
            WHERE Dno = OLD.Dno;
        END;

R4: CREATE TRIGGER Total_sal4
        AFTER DELETE ON EMPLOYEE
        FOR EACH ROW
        WHEN ( OLD.Dno IS NOT NULL )
            UPDATE DEPARTMENT
            SET Total_sal = Total_sal - OLD.Salary
            WHERE Dno = OLD.Dno;

(b) R5: CREATE TRIGGER Inform_supervisor1
        BEFORE INSERT OR UPDATE OF Salary, Supervisor_ssn
        ON EMPLOYEE
        FOR EACH ROW
        WHEN ( NEW.Salary > ( SELECT Salary FROM EMPLOYEE
                               WHERE Ssn = NEW.Supervisor_ssn ) )
            inform_supervisor(NEW.Supervisor_ssn, NEW.Ssn );
```



action is executed whenever the triggering event occurs. The action updates both the old department and new department of the reassigned employees by adding their salary to `Total_sal` of their *new* department and subtracting their salary from `Total_sal` of their *old* department. Note that this should work even if the value of `Dno` is `NULL`, because in this case no department will be selected for the rule action.<sup>5</sup>

It is important to note the effect of the optional `FOR EACH ROW` clause, which signifies that the rule is triggered separately *for each tuple*. This is known as a **row-level trigger**. If this clause was left out, the trigger would be known as a **statement-level trigger** and would be triggered once for each triggering statement. To see the difference, consider the following update operation, which gives a 10% raise to all employees assigned to department 5. This operation would be an event that triggers rule R2:

```
UPDATE EMPLOYEE
SET      Salary = 1.1 * Salary
WHERE    Dno = 5;
```

Because the above statement could update multiple records, a rule using row-level semantics, such as R2 in Figure 26.2, would be triggered *once for each row*, whereas a rule using statement-level semantics is triggered *only once*. The Oracle system allows the user to choose which of the above options is to be used for each rule. Including the optional `FOR EACH ROW` clause creates a row-level trigger, and leaving it out creates a statement-level trigger. Note that the keywords `NEW` and `OLD` can only be used with row-level triggers.

As a second example, suppose we want to check whenever an employee's salary is greater than the salary of his or her direct supervisor. Several events can trigger this rule: inserting a new employee, changing an employee's salary, or changing an employee's supervisor. Suppose that the action to take would be to call an external procedure `inform_supervisor`,<sup>6</sup> which will notify the supervisor. The rule could then be written as in R5 (see Figure 26.2(b)).

Figure 26.3 shows the syntax for specifying some of the main options available in Oracle triggers. We will describe the syntax for triggers in the SQL-99 standard in Section 26.1.5.

### 26.1.2 Design and Implementation Issues for Active Databases

The previous section gave an overview of some of the main concepts for specifying active rules. In this section, we discuss some additional issues concerning how rules are designed and implemented. The first issue concerns activation,

---

<sup>5</sup>R1, R2, and R4 can also be written without a condition. However, it may be more efficient to execute them with the condition since the action is not invoked unless it is required.

<sup>6</sup>Assuming that an appropriate external procedure has been declared. This is a feature that is available in SQL-99 and later standards.

```

<trigger> ::= CREATE TRIGGER <trigger name>
           ( AFTER | BEFORE ) <triggering events> ON <table name>
           [ FOR EACH ROW ]
           [ WHEN <condition> ]
           <trigger actions> ;
<triggering events> ::= <trigger event> {OR <trigger event> }
<trigger event> ::= INSERT | DELETE | UPDATE [ OF <column name> { , <column name> } ]
<trigger action> ::= <PL/SQL block>

```

**Figure 26.3**

A syntax summary for specifying triggers in the Oracle system (main options only).

---

deactivation, and grouping of rules. In addition to creating rules, an active database system should allow users to *activate*, *deactivate*, and *drop* rules by referring to their rule names. A **deactivated rule** will not be triggered by the triggering event. This feature allows users to selectively deactivate rules for certain periods of time when they are not needed. The **activate command** will make the rule active again. The **drop command** deletes the rule from the system. Another option is to group rules into named **rule sets**, so the whole set of rules can be activated, deactivated, or dropped. It is also useful to have a command that can trigger a rule or rule set via an explicit **PROCESS RULES** command issued by the user.

The second issue concerns whether the triggered action should be executed *before*, *after*, *instead of*, or *concurrently with* the triggering event. A **before trigger** executes the trigger before executing the event that caused the trigger. It can be used in applications such as checking for constraint violations. An **after trigger** executes the trigger after executing the event, and it can be used in applications such as maintaining derived data and monitoring for specific events and conditions. An **instead of trigger** executes the trigger instead of executing the event, and it can be used in applications such as executing corresponding updates on base relations in response to an event that is an update of a view.

A related issue is whether the action being executed should be considered as a *separate transaction* or whether it should be part of the same transaction that triggered the rule. We will try to categorize the various options. It is important to note that not all options may be available for a particular active database system. In fact, most commercial systems are *limited to one or two of the options* that we will now discuss.

Let us assume that the triggering event occurs as part of a transaction execution. We should first consider the various options for how the triggering event is related to the evaluation of the rule's condition. The rule *condition evaluation* is also known as **rule consideration**, since the action is to be executed only after considering whether the condition evaluates to true or false. There are three main possibilities for rule consideration:

1. **Immediate consideration.** The condition is evaluated as part of the same transaction as the triggering event and is evaluated *immediately*. This case can be further categorized into three options:
  - Evaluate the condition *before* executing the triggering event.
  - Evaluate the condition *after* executing the triggering event.
  - Evaluate the condition *instead of* executing the triggering event.
2. **Deferred consideration.** The condition is evaluated at the end of the transaction that included the triggering event. In this case, there could be many triggered rules waiting to have their conditions evaluated.
3. **Detached consideration.** The condition is evaluated as a separate transaction, spawned from the triggering transaction.

The next set of options concerns the relationship between evaluating the rule condition and *executing* the rule action. Here, again, three options are possible: **immediate**, **deferred**, or **detached** execution. Most active systems use the first option. That is, as soon as the condition is evaluated, if it returns true, the action is *immediately* executed.

The Oracle system (see Section 26.1.1) uses the *immediate consideration* model, but it allows the user to specify for each rule whether the *before* or *after* option is to be used with immediate condition evaluation. It also uses the *immediate execution* model. The STARBURST system (see Section 26.1.3) uses the *deferred consideration* option, meaning that all rules triggered by a transaction wait until the triggering transaction reaches its end and issues its COMMIT WORK command before the rule conditions are evaluated.<sup>7</sup>

Another issue concerning active database rules is the distinction between *row-level rules* and *statement-level rules*. Because SQL update statements (which act as triggering events) can specify a set of tuples, one must distinguish between whether the rule should be considered once for the *whole statement* or whether it should be considered separately *for each row* (that is, tuple) affected by the statement. The SQL-99 standard (see Section 26.1.5) and the Oracle system (see Section 26.1.1) allow the user to choose which of the options is to be used for each rule, whereas STARBURST uses statement-level semantics only. We will give examples of how statement-level triggers can be specified in Section 26.1.3.

One of the difficulties that may have limited the widespread use of active rules, in spite of their potential to simplify database and software development, is that there are no easy-to-use techniques for designing, writing, and verifying rules. For example, it is difficult to verify that a set of rules is **consistent**, meaning that two or more rules in the set do not contradict one another. It is also difficult to guarantee **termination** of a set of rules under all circumstances. To illustrate the termination problem briefly, consider the rules in Figure 26.4. Here, rule R1 is triggered by an INSERT event on TABLE1 and its action includes an update event on Attribute1 of

---

<sup>7</sup>STARBURST also allows the user to start rule consideration explicitly via a PROCESS RULES command.

```

R1: CREATE TRIGGER T1
    AFTER INSERT ON TABLE1
    FOR EACH ROW
    UPDATE TABLE2
    SET Attribute1 = ... ;

R2: CREATE TRIGGER T2
    AFTER UPDATE OF Attribute1 ON TABLE2
    FOR EACH ROW
    INSERT INTO TABLE1 VALUES ( ... );

```

**Figure 26.4**

An example to illustrate the termination problem for active rules.

TABLE2. However, rule R2's triggering event is an UPDATE event on Attribute1 of TABLE2, and its action includes an INSERT event on TABLE1. In this example, it is easy to see that these two rules can trigger one another indefinitely, leading to non-termination. However, if dozens of rules are written, it is very difficult to determine whether termination is guaranteed or not.

If active rules are to reach their potential, it is necessary to develop tools for the design, debugging, and monitoring of active rules that can help users design and debug their rules.

### 26.1.3 Examples of Statement-Level Active Rules in STARBURST

We now give some examples to illustrate how rules can be specified in the STARBURST experimental DBMS. This will allow us to demonstrate how statement-level rules can be written, since these are the only types of rules allowed in STARBURST.

The three active rules R1S, R2S, and R3S in Figure 26.5 correspond to the first three rules in Figure 26.2, but they use STARBURST notation and statement-level semantics. We can explain the rule structure using rule R1S. The CREATE RULE statement specifies a rule name—Total\_sal1 for R1S. The ON clause specifies the relation on which the rule is specified—EMPLOYEE for R1S. The WHEN clause is used to specify the **events** that trigger the rule.<sup>8</sup> The *optional* IF clause is used to specify any **conditions** that need to be checked. Finally, the THEN clause is used to specify the **actions** to be taken, which are typically one or more SQL statements.

In STARBURST, the basic events that can be specified for triggering the rules are the standard SQL update commands: INSERT, DELETE, and UPDATE. These are specified by the keywords **INSERTED**, **DELETED**, and **UPDATED** in STARBURST notation. Second, the rule designer needs to have a way to refer to the tuples that have been modified. The keywords **INSERTED**, **DELETED**, **NEW-UPDATED**, and **OLD-UPDATED** are used in STARBURST notation to refer to four **transition tables** (relations) that include the newly inserted tuples, the deleted tuples, the updated

<sup>8</sup>Note that the WHEN keyword specifies *events* in STARBURST but is used to specify the rule *condition* in SQL and Oracle triggers.

```

R1S: CREATE RULE Total_sal1 ON EMPLOYEE
      WHEN INSERTED
      IF EXISTS ( SELECT * FROM INSERTED WHERE Dno IS NOT NULL)
      THEN UPDATE DEPARTMENT AS D
               SET D.Total_sal = D.Total_sal +
                   ( SELECT SUM (I.Salary) FROM INSERTED AS I WHERE D.Dno = I.Dno )
               WHERE D.Dno IN ( SELECT Dno FROM INSERTED );

R2S: CREATE RULE Total_sal2 ON EMPLOYEE
      WHEN UPDATED ( Salary )
      IF EXISTS ( SELECT * FROM NEW-UPDATED WHERE Dno IS NOT NULL)
      OR EXISTS ( SELECT * FROM OLD-UPDATED WHERE Dno IS NOT NULL)
      THEN UPDATE DEPARTMENT AS D
               SET D.Total_sal = D.Total_sal +
                   ( SELECT SUM (N.Salary) FROM NEW-UPDATED AS N
                     WHERE D.Dno = N.Dno ) -
                   ( SELECT SUM (O.Salary) FROM OLD-UPDATED AS O
                     WHERE D.Dno = O.Dno )
               WHERE D.Dno IN ( SELECT Dno FROM NEW-UPDATED ) OR
                   D.Dno IN ( SELECT Dno FROM OLD-UPDATED );

R3S: CREATE RULE Total_sal3 ON EMPLOYEE
      WHEN UPDATED ( Dno )
      THEN UPDATE DEPARTMENT AS D
               SET D.Total_sal = D.Total_sal +
                   ( SELECT SUM (N.Salary) FROM NEW-UPDATED AS N
                     WHERE D.Dno = N.Dno )
               WHERE D.Dno IN ( SELECT Dno FROM NEW-UPDATED );
      UPDATE DEPARTMENT AS D
      SET D.Total_sal = Total_sal -
          ( SELECT SUM (O.Salary) FROM OLD-UPDATED AS O
            WHERE D.Dno = O.Dno )
      WHERE D.Dno IN ( SELECT Dno FROM OLD-UPDATED );

```

**Figure 26.5**

Active rules using statement-level semantics in STARBURST notation.

tuples *before* they were updated, and the updated tuples *after* they were updated, respectively. Obviously, depending on the triggering events, only some of these transition tables may be available. The rule writer can refer to these tables when writing the condition and action parts of the rule. Transition tables contain tuples of the same type as those in the relation specified in the ON clause of the rule—for R1S, R2S, and R3S, this is the EMPLOYEE relation.

In statement-level semantics, the rule designer can only refer to the transition tables as a whole and the rule is triggered only once, so the rules must be written differently than for row-level semantics. Because multiple employee tuples may be

inserted in a single insert statement, we have to check if *at least one* of the newly inserted employee tuples is related to a department. In R1S, the condition

**EXISTS (SELECT \* FROM INSERTED WHERE Dno IS NOT NULL )**

is checked, and if it evaluates to true, then the action is executed. The action updates in a single statement the DEPARTMENT tuple(s) related to the newly inserted employee(s) by adding their salaries to the Total\_sal attribute of each related department. Because more than one newly inserted employee may belong to the same department, we use the SUM aggregate function to ensure that all their salaries are added.

Rule R2S is similar to R1S, but is triggered by an UPDATE operation that updates the salary of one or more employees rather than by an INSERT. Rule R3S is triggered by an update to the Dno attribute of EMPLOYEE, which signifies changing one or more employees' assignment from one department to another. There is no condition in R3S, so the action is executed whenever the triggering event occurs.<sup>9</sup> The action updates both the old department(s) and new department(s) of the reassigned employees by adding their salary to Total\_sal of each *new* department and subtracting their salary from Total\_sal of each *old* department.

In our example, it is more complex to write the statement-level rules than the row-level rules, as can be illustrated by comparing Figures 26.2 and 26.5. However, this is not a general rule, and other types of active rules may be easier to specify when using statement-level notation than when using row-level notation.

The execution model for active rules in STARBURST uses **deferred consideration**. That is, all the rules that are triggered within a transaction are placed in a set—called the **conflict set**—which is not considered for evaluation of conditions and execution until the transaction ends (by issuing its COMMIT WORK command). STARBURST also allows the user to explicitly start rule consideration in the middle of a transaction via an explicit PROCESS RULES command. Because multiple rules must be evaluated, it is necessary to specify an order among the rules. The syntax for rule declaration in STARBURST allows the specification of *ordering* among the rules to instruct the system about the order in which a set of rules should be considered.<sup>10</sup> Additionally, the transition tables—INSERTED, DELETED, NEW-UPDATED, and OLD-UPDATED—contain the *net effect* of all the operations within the transaction that affected each table, since multiple operations may have been applied to each table during the transaction.

### 26.1.4 Potential Applications for Active Databases

We now briefly discuss some of the potential applications of active rules. Obviously, one important application is to allow **notification** of certain conditions that

<sup>9</sup>As in the Oracle examples, rules R1S and R2S can be written without a condition. However, it may be more efficient to execute them with the condition since the action is not invoked unless it is required.

<sup>10</sup>If no order is specified between a pair of rules, the system default order is based on placing the rule declared first ahead of the other rule.

occur. For example, an active database may be used to monitor, say, the temperature of an industrial furnace. The application can periodically insert in the database the temperature reading records directly from temperature sensors, and active rules can be written that are triggered whenever a temperature record is inserted, with a condition that checks if the temperature exceeds the danger level and results in the action to raise an alarm.

Active rules can also be used to **enforce integrity constraints** by specifying the types of events that may cause the constraints to be violated and then evaluating appropriate conditions that check whether the constraints are actually violated by the event or not. Hence, complex application constraints, often known as **business rules**, may be enforced that way. For example, in the UNIVERSITY database application, one rule may monitor the GPA of students whenever a new grade is entered, and it may alert the advisor if the GPA of a student falls below a certain threshold; another rule may check that course prerequisites are satisfied before allowing a student to enroll in a course; and so on.

Other applications include the automatic **maintenance of derived data**, such as the examples of rules R1 through R4 that maintain the derived attribute `Total_sal` whenever individual employee tuples are changed. A similar application is to use active rules to maintain the consistency of **materialized views** (see Section 5.3) whenever the base relations are modified. Alternately, an update operation specified on a view can be a triggering event, which can be converted to updates on the base relations by using an *instead of* trigger. These applications are also relevant to the new data warehousing technologies (see Chapter 29). A related application maintains that **replicated tables** are consistent by specifying rules that modify the replicas whenever the master table is modified.

### 26.1.5 Triggers in SQL-99

Triggers in the SQL-99 and later standards are similar to the examples we discussed in Section 26.1.1, with some minor syntactic differences. The basic **events** that can be specified for triggering the rules are the standard SQL update commands: INSERT, DELETE, and UPDATE. In the case of UPDATE, one may specify the attributes to be updated. Both row-level and statement-level triggers are allowed, indicated in the trigger by the clauses FOR EACH ROW and FOR EACH STATEMENT, respectively. One syntactic difference is that the trigger may specify particular tuple variable names for the old and new tuples instead of using the keywords NEW and OLD, as shown in Figure 26.1. Trigger T1 in Figure 26.6 shows how the row-level trigger R2 from Figure 26.1(a) may be specified in SQL-99. Inside the REFERENCING clause, we named tuple variables (aliases) O and N to refer to the OLD tuple (before modification) and NEW tuple (after modification), respectively. Trigger T2 in Figure 26.6 shows how the statement-level trigger R2S from Figure 26.5 may be specified in SQL-99. For a statement-level trigger, the REFERENCING clause is used to refer to the table of all new tuples (newly inserted or newly updated) as N, whereas the table of all old tuples (deleted tuples or tuples before they were updated) is referred to as O.



```

T1: CREATE TRIGGER Total_sal1
AFTER UPDATE OF Salary ON EMPLOYEE
REFERENCING OLD ROW AS O, NEW ROW AS N
FOR EACH ROW
WHEN ( N.Dno IS NOT NULL )
UPDATE DEPARTMENT
SET Total_sal = Total_sal + N.salary - O.salary
WHERE Dno = N.Dno;

T2: CREATE TRIGGER Total_sal2
AFTER UPDATE OF Salary ON EMPLOYEE
REFERENCING OLD TABLE AS O, NEW TABLE AS N
FOR EACH STATEMENT
WHEN EXISTS ( SELECT *FROM N WHERE N.Dno IS NOT NULL ) OR
EXISTS ( SELECT * FROM O WHERE O.Dno IS NOT NULL )
UPDATE DEPARTMENT AS D
SET D.Total_sal = D.Total_sal
+ ( SELECT SUM (N.Salary) FROM N WHERE D.Dno=N.Dno )
- ( SELECT SUM (O.Salary) FROM O WHERE D.Dno=O.Dno )
WHERE Dno IN ( ( SELECT Dno FROM N ) UNION ( SELECT Dno FROM O ) );

```

**Figure 26.6**

Trigger T1 illustrating the syntax for defining triggers in SQL-99.

## 26.2 Temporal Database Concepts

Temporal databases, in the broadest sense, encompass all database applications that require some aspect of time when organizing their information. Hence, they provide a good example to illustrate the need for developing a set of unifying concepts for application developers to use. Temporal database applications have been developed since the early days of database usage. However, in creating these applications, it is mainly left to the application designers and developers to discover, design, program, and implement the temporal concepts they need. There are many examples of applications where some aspect of time is needed to maintain the information in a database. These include *healthcare*, where patient histories need to be maintained; *insurance*, where claims and accident histories are required as well as information about the times when insurance policies are in effect; *reservation systems* in general (hotel, airline, car rental, train, and so on), where information on the dates and times when reservations are in effect are required; *scientific databases*, where data collected from experiments includes the time when each data is measured; and so on. Even the two examples used in this book may be easily expanded into temporal applications. In the COMPANY database, we may wish to keep SALARY, JOB, and PROJECT histories on each employee. In the UNIVERSITY database, time is already included in the SEMESTER and YEAR of each SECTION of a COURSE, the grade history of a STUDENT, and the information on research grants. In fact, it is realistic to conclude that the majority of database applications have some temporal information. However, users often attempt to simplify or ignore temporal aspects because of the complexity that they add to their applications.



In this section, we will introduce some of the concepts that have been developed to deal with the complexity of temporal database applications. Section 26.2.1 gives an overview of how time is represented in databases, the different types of temporal information, and some of the different dimensions of time that may be needed. Section 26.2.2 discusses how time can be incorporated into relational databases. Section 26.2.3 gives some additional options for representing time that are possible in database models that allow complex-structured objects, such as object databases. Section 26.2.4 introduces operations for querying temporal databases and gives a brief overview of the TSQL2 language, which extends SQL with temporal concepts. Section 26.2.5 focuses on time series data, which is a type of temporal data that is very important in practice.

### 26.2.1 Time Representation, Calendars, and Time Dimensions

For temporal databases, time is considered to be an *ordered sequence* of **points** in some **granularity** that is determined by the application. For example, suppose that some temporal application never requires time units that are less than one second. Then, each time point represents one second using this granularity. In reality, each second is a (short) *time duration*, not a point, since it may be further divided into milliseconds, microseconds, and so on. Temporal database researchers have used the term **chronon** instead of *point* to describe this minimal granularity for a particular application. The main consequence of choosing a minimum granularity—say, one second—is that events occurring within the same second will be considered to be *simultaneous events*, even though in reality they may not be.

Because there is no known beginning or ending of time, one needs a reference point from which to measure specific time points. Various calendars are used by various cultures (such as Gregorian (Western), Chinese, Islamic, Hindu, Jewish, Coptic, and so on) with different reference points. A **calendar** organizes time into different time units for convenience. Most calendars group 60 seconds into a minute, 60 minutes into an hour, 24 hours into a day (based on the physical time of earth's rotation around its axis), and 7 days into a week. Further groupings of days into months and months into years either follow solar or lunar natural phenomena and are generally irregular. In the Gregorian calendar, which is used in most Western countries, days are grouped into months that are 28, 29, 30, or 31 days, and 12 months are grouped into a year. Complex formulas are used to map the different time units to one another.

In SQL2, the temporal data types (see Chapter 4) include DATE (specifying Year, Month, and Day as YYYY-MM-DD), TIME (specifying Hour, Minute, and Second as HH:MM:SS), TIMESTAMP (specifying a Date/Time combination, with options for including subsecond divisions if they are needed), INTERVAL (a relative time duration, such as 10 days or 250 minutes), and PERIOD (an *anchored* time duration with a fixed starting point, such as the 10-day period from January 1, 2009, to January 10, 2009, inclusive).<sup>11</sup>

---

<sup>11</sup>Unfortunately, the terminology has not been used consistently. For example, the term *interval* is often used to denote an anchored duration. For consistency, we will use the SQL terminology.

**Event Information versus Duration (or State) Information.** A temporal database will store information concerning when certain events occur, or when certain facts are considered to be true. There are several different types of temporal information. **Point events** or **facts** are typically associated in the database with a **single time point** in some granularity. For example, a bank deposit event may be associated with the timestamp when the deposit was made, or the total monthly sales of a product (fact) may be associated with a particular month (say, February 2010). Note that even though such events or facts may have different granularities, each is still associated with a *single time value* in the database. This type of information is often represented as **time series data**, as we will discuss in Section 26.2.5. **Duration events** or **facts**, on the other hand, are associated with a specific **time period** in the database.<sup>12</sup> For example, an employee may have worked in a company from August 15, 2003 until November 20, 2008.

A **time period** is represented by its **start** and **end time points** [START-TIME, ENDTIME]. For example, the above period is represented as [2003-08-15, 2008-11-20]. Such a time period is often interpreted to mean the *set of all time points* from start-time to end-time, inclusive, in the specified granularity. Hence, assuming day granularity, the period [2003-08-15, 2008-11-20] represents the set of all days from August 15, 2003, until November 20, 2008, inclusive.<sup>13</sup>

**Valid Time and Transaction Time Dimensions.** Given a particular event or fact that is associated with a particular time point or time period in the database, the association may be interpreted to mean different things. The most natural interpretation is that the associated time is the time that the event occurred, or the period during which the fact was considered to be true *in the real world*. If this interpretation is used, the associated time is often referred to as the **valid time**. A temporal database using this interpretation is called a **valid time database**.

However, a different interpretation can be used, where the associated time refers to the time when the information was actually stored in the database; that is, it is the value of the system time clock when the information is valid *in the system*.<sup>14</sup> In this case, the associated time is called the **transaction time**. A temporal database using this interpretation is called a **transaction time database**.

Other interpretations can also be intended, but these are considered to be the most common ones, and they are referred to as **time dimensions**. In some applications, only one of the dimensions is needed and in other cases both time dimensions are required, in which case the temporal database is called a **bitemporal database**. If

<sup>12</sup>This is the same as an *anchored duration*. It has also been frequently called a *time interval*, but to avoid confusion we will use *period* to be consistent with SQL terminology.

<sup>13</sup>The representation [2003-08-15, 2008-11-20] is called a *closed interval* representation. One can also use an *open interval*, denoted [2003-08-15, 2008-11-21), where the set of points does not include the end point. Although the latter representation is sometimes more convenient, we shall use closed intervals except where indicated.

<sup>14</sup>The explanation is more involved, as we will see in Section 26.2.3.

other interpretations are intended for time, the user can define the semantics and program the applications appropriately, and this interpretation of time is called a **user-defined time**.

The next section shows how these concepts can be incorporated into relational databases, and Section 26.2.3 shows an approach to incorporate temporal concepts into object databases.

### 26.2.2 Incorporating Time in Relational Databases Using Tuple Versioning

**Valid Time Relations.** Let us now see how the different types of temporal databases may be represented in the relational model. First, suppose that we would like to include the history of changes as they occur in the real world. Consider again the database in Figure 26.1, and let us assume that, for this application, the granularity is day. Then, we could convert the two relations EMPLOYEE and DEPARTMENT into **valid time relations** by adding the attributes Vst (Valid Start Time) and Vet (Valid End Time), whose data type is DATE in order to provide day granularity. This is shown in Figure 26.7(a), where the relations have been renamed EMP\_VT and DEPT\_VT, respectively.

Consider how the EMP\_VT relation differs from the nontemporal EMPLOYEE relation (Figure 26.1).<sup>15</sup> In EMP\_VT, each tuple V represents a **version** of an employee's

(a) EMP\_VT

Name	<u>Ssn</u>	Salary	Dno	Supervisor_ssn	<u>Vst</u>	Vet
------	------------	--------	-----	----------------	------------	-----

DEPT\_VT

Dname	<u>Dno</u>	Total_sal	Manager_ssn	<u>Vst</u>	Vet
-------	------------	-----------	-------------	------------	-----

(b) EMP\_TT

Name	<u>Ssn</u>	Salary	Dno	Supervisor_ssn	<u>Tst</u>	Tet
------	------------	--------	-----	----------------	------------	-----

DEPT\_TT

Dname	<u>Dno</u>	Total_sal	Manager_ssn	<u>Tst</u>	Tet
-------	------------	-----------	-------------	------------	-----

(c) EMP\_BT

Name	<u>Ssn</u>	Salary	Dno	Supervisor_ssn	<u>Vst</u>	Vet	<u>Tst</u>	Tet
------	------------	--------	-----	----------------	------------	-----	------------	-----

DEPT\_BT

Dname	<u>Dno</u>	Total_sal	Manager_ssn	<u>Vst</u>	Vet	<u>Tst</u>	Tet
-------	------------	-----------	-------------	------------	-----	------------	-----

**Figure 26.7**

Different types of temporal relational databases. (a) Valid time database schema. (b) Transaction time database schema. (c) Bitemporal database schema.

<sup>15</sup>A nontemporal relation is also called a **snapshot relation** because it shows only the *current snapshot* or *current state* of the database.

information that is valid (in the real world) only during the time period [V.Vst, V.Vet], whereas in EMPLOYEE each tuple represents only the current state or current version of each employee. In EMP\_VT, the **current version** of each employee typically has a special value, *now*, as its valid end time. This special value, *now*, is a **temporal variable** that implicitly represents the current time as time progresses. The nontemporal EMPLOYEE relation would only include those tuples from the EMP\_VT relation whose Vet is *now*.

Figure 26.8 shows a few tuple versions in the valid-time relations EMP\_VT and DEPT\_VT. There are two versions of Smith, three versions of Wong, one version of Brown, and one version of Narayan. We can now see how a valid time relation should behave when information is changed. Whenever one or more attributes of an employee are **updated**, rather than actually overwriting the old values, as would happen in a nontemporal relation, the system should create a new version and **close** the current version by changing its Vet to the end time. Hence, when the user issued the command to update the salary of Smith effective on June 1, 2003, to \$30000, the second version of Smith was created (see Figure 26.8). At the time of this update, the first version of Smith was the current version, with *now* as its Vet, but after the update *now* was changed to May 31, 2003 (one less than June 1, 2003, in day granularity), to indicate that the version has become a **closed** or **history version** and that the new (second) version of Smith is now the current one.

It is important to note that in a valid time relation, the user must generally provide the valid time of an update. For example, the salary update of Smith may have been

**Figure 26.8**

Some tuple versions in the valid time relations EMP\_VT and DEPT\_VT.

#### EMP\_VT

Name	Ssn	Salary	Dno	Supervisor_ssn	Vst	Vet
Smith	123456789	25000	5	333445555	2002-06-15	2003-05-31
Smith	123456789	30000	5	333445555	2003-06-01	Now
Wong	333445555	25000	4	999887777	1999-08-20	2001-01-31
Wong	333445555	30000	5	999887777	2001-02-01	2002-03-31
Wong	333445555	40000	5	888665555	2002-04-01	Now
Brown	222447777	28000	4	999887777	2001-05-01	2002-08-10
Narayan	666884444	38000	5	333445555	2003-08-01	Now

...

#### DEPT\_VT

Dname	Dno	Manager_ssn	Vst	Vet
Research	5	888665555	2001-09-20	2002-03-31
Research	5	333445555	2002-04-01	Now

...

entered in the database on May 15, 2003, at 8:52:12 a.m., say, even though the salary change in the real world is effective on June 1, 2003. This is called a **proactive update**, since it is applied to the database *before* it becomes effective in the real world. If the update is applied to the database *after* it becomes effective in the real world, it is called a **retroactive update**. An update that is applied at the same time as it becomes effective is called a **simultaneous update**.

The action that corresponds to **deleting** an employee in a nontemporal database would typically be applied to a valid time database by *closing the current version* of the employee being deleted. For example, if Smith leaves the company effective January 19, 2004, then this would be applied by changing Vet of the current version of Smith from *now* to 2004-01-19. In Figure 26.8, there is no current version for Brown, because he presumably left the company on 2002-08-10 and was *logically deleted*. However, because the database is temporal, the old information on Brown is still there.

The operation to **insert** a new employee would correspond to *creating the first tuple version* for that employee and making it the current version, with the Vst being the effective (real world) time when the employee starts work. In Figure 26.7, the tuple on Narayan illustrates this, since the first version has not been updated yet.

Notice that in a valid time relation, the *nontemporal key*, such as Ssn in EMPLOYEE, is no longer unique in each tuple (version). The new relation key for EMP\_VT is a combination of the nontemporal key and the valid start time attribute Vst,<sup>16</sup> so we use (Ssn, Vst) as primary key. This is because, at any point in time, there should be *at most one valid version* of each entity. Hence, the constraint that any two tuple versions representing the same entity should have *nonintersecting valid time periods* should hold on valid time relations. Notice that if the nontemporal primary key value may change over time, it is important to have a unique **surrogate key attribute**, whose value never changes for each real-world entity, in order to relate all versions of the same real-world entity.

Valid time relations basically keep track of the history of changes as they become effective in the *real world*. Hence, if all real-world changes are applied, the database keeps a history of the *real-world states* that are represented. However, because updates, insertions, and deletions may be applied retroactively or proactively, there is no record of the actual *database state* at any point in time. If the actual database states are important to an application, then one should use *transaction time relations*.

**Transaction Time Relations.** In a transaction time database, whenever a change is applied to the database, the actual **timestamp** of the transaction that applied the change (insert, delete, or update) is recorded. Such a database is most useful when changes are applied *simultaneously* in the majority of cases—for example, real-time stock trading or banking transactions. If we convert the nontemporal database in Figure 26.1 into a transaction time database, then the two relations EMPLOYEE and DEPARTMENT are converted into **transaction time relations** by adding the attributes Tst (Transaction Start Time) and Tet (Transaction End Time), whose data

<sup>16</sup>A combination of the nontemporal key and the valid end time attribute **Vet** could also be used.

type is typically **TIMESTAMP**. This is shown in Figure 26.7(b), where the relations have been renamed **EMP\_TT** and **DEPT\_TT**, respectively.

In **EMP\_TT**, each tuple  $V$  represents a *version* of an employee's information that was created at actual time  $V.Tst$  and was (logically) removed at actual time  $V.Tet$  (because the information was no longer correct). In **EMP\_TT**, the *current version* of each employee typically has a special value, **uc (Until Changed)**, as its transaction end time, which indicates that the tuple represents correct information *until it is changed* by some other transaction.<sup>17</sup> A transaction time database has also been called a **rollback database**,<sup>18</sup> because a user can logically roll back to the actual database state at any past point in time  $T$  by retrieving all tuple versions  $V$  whose transaction time period  $[V.Tst, V.Tet]$  includes time point  $T$ .

**Bitemporal Relations.** Some applications require both valid time and transaction time, leading to **bitemporal relations**. In our example, Figure 26.7(c) shows how the **EMPLOYEE** and **DEPARTMENT** nontemporal relations in Figure 26.1 would appear as bitemporal relations **EMP\_BT** and **DEPT\_BT**, respectively. Figure 26.9 shows a few tuples in these relations. In these tables, tuples whose transaction end time  $Tet$  is **uc** are the ones representing currently valid information, whereas tuples whose  $Tet$  is an absolute timestamp are tuples that were valid until (just before) that timestamp. Hence, the tuples with **uc** in Figure 26.9 correspond to the valid time tuples in Figure 26.7. The transaction start time attribute  $Tst$  in each tuple is the timestamp of the transaction that created that tuple.

Now consider how an **update operation** would be implemented on a bitemporal relation. In this model of bitemporal databases,<sup>19</sup> *no attributes are physically changed* in any tuple except for the transaction end time attribute  $Tet$  with a value of **uc**.<sup>20</sup> To illustrate how tuples are created, consider the **EMP\_BT** relation. The *current version*  $V$  of an employee has **uc** in its  $Tet$  attribute and *now* in its  $Vet$  attribute. If some attribute—say, **Salary**—is updated, then the transaction  $T$  that performs the update should have two parameters: the new value of **Salary** and the valid time  $VT$  when the new salary becomes effective (in the real world). Assume that  $VT-$  is the time point before  $VT$  in the given valid time granularity and that transaction  $T$  has a timestamp  $TS(T)$ . Then, the following physical changes would be applied to the **EMP\_BT** table:

1. Make a copy  $V_2$  of the current version  $V$ ; set  $V_2.Vet$  to  $VT-$ ,  $V_2.Tst$  to  $TS(T)$ ,  $V_2.Tet$  to **uc**, and insert  $V_2$  in **EMP\_BT**;  $V_2$  is a copy of the previous current version  $V$  *after it is closed* at valid time  $VT-$ .

<sup>17</sup>The **uc** variable in transaction time relations corresponds to the *now* variable in valid time relations. However, the semantics are slightly different.

<sup>18</sup>Here, the term *rollback* does not have the same meaning as *transaction rollback* (see Chapter 23) during recovery, where the transaction updates are *physically undone*. Rather, here the updates can be *logically undone*, allowing the user to examine the database as it appeared at a previous time point.

<sup>19</sup>There have been many proposed temporal database models. We describe specific models here as examples to illustrate the concepts.

<sup>20</sup>Some bitemporal models allow the  $Vet$  attribute to be changed also, but the interpretations of the tuples are different in those models.

EMP\_BT

Name	Ssn	Salary	Dno	Supervisor_ssn	Vst	Vet	Tst	Tet
Smith	123456789	25000	5	333445555	2002-06-15	Now	2002-06-08, 13:05:58	2003-06-04,08:56:12
Smith	123456789	25000	5	333445555	2002-06-15	2003-05-31	2003-06-04, 08:56:12	uc
Smith	123456789	30000	5	333445555	2003-06-01	Now	2003-06-04, 08:56:12	uc
Wong	333445555	25000	4	999887777	1999-08-20	Now	1999-08-20, 11:18:23	2001-01-07,14:33:02
Wong	333445555	25000	4	999887777	1999-08-20	2001-01-31	2001-01-07, 14:33:02	uc
Wong	333445555	30000	5	999887777	2001-02-01	Now	2001-01-07, 14:33:02	2002-03-28,09:23:57
Wong	333445555	30000	5	999887777	2001-02-01	2002-03-31	2002-03-28, 09:23:57	uc
Wong	333445555	40000	5	888667777	2002-04-01	Now	2002-03-28, 09:23:57	uc
Brown	222447777	28000	4	999887777	2001-05-01	Now	2001-04-27, 16:22:05	2002-08-12,10:11:07
Brown	222447777	28000	4	999887777	2001-05-01	2002-08-10	2002-08-12, 10:11:07	uc
Narayan	666884444	38000	5	333445555	2003-08-01	Now	2003-07-28, 09:25:37	uc

...

DEPT\_VT

Dname	Dno	Manager_ssn	Vst	Vet	Tst	Tet
Research	5	888665555	2001-09-20	Now	2001-09-15,14:52:12	2001-03-28,09:23:57
Research	5	888665555	2001-09-20	1997-03-31	2002-03-28,09:23:57	uc
Research	5	333445555	2002-04-01	Now	2002-03-28,09:23:57	uc

**Figure 26.9**

Some tuple versions in the bitemporal relations EMP\_BT and DEPT\_BT.

2. Make a copy  $V_3$  of the current version  $V$ ; set  $V_3.Vst$  to  $VT$ ,  $V_3.Vet$  to *now*,  $V_3.Salary$  to the new salary value,  $V_3.Tst$  to  $TS(T)$ ,  $V_3.Tet$  to *uc*, and insert  $V_3$  in EMP\_BT;  $V_3$  represents the new current version.
3. Set  $V.Tet$  to  $TS(T)$  since the current version is no longer representing correct information.

As an illustration, consider the first three tuples  $V_1$ ,  $V_2$ , and  $V_3$  in EMP\_BT in Figure 26.9. Before the update of Smith's salary from 25000 to 30000, only  $V_1$  was in EMP\_BT and it was the current version and its Tet was *uc*. Then, a transaction  $T$  whose timestamp  $TS(T)$  is '2003-06-04,08:56:12' updates the salary to 30000 with the effective valid time of '2003-06-01'. The tuple  $V_2$  is created, which is a copy of  $V_1$  except that its Vet is set to '2003-05-31', one day less than the new valid time, and its Tst is the timestamp of the updating transaction. The tuple  $V_3$  is also created, which has the new salary, its Vst is set to '2003-06-01', and its Tst is also the timestamp of the updating transaction. Finally, the Tet of  $V_1$  is set to the timestamp of the updating transaction, '2003-06-04,08:56:12'. Note that this is a *retroactive update*, since the updating transaction ran on June 4, 2003, but the salary change is effective on June 1, 2003.

Similarly, when Wong's salary and department are updated (at the same time) to 30000 and 5, the updating transaction's timestamp is '2001-01-07,14:33:02' and the effective valid time for the update is '2001-02-01'. Hence, this is a *proactive update* because the transaction ran on January 7, 2001, but the effective date was February 1, 2001. In this case, tuple  $V_4$  is logically replaced by  $V_5$  and  $V_6$ .



Next, let us illustrate how a **delete operation** would be implemented on a bitemporal relation by considering the tuples  $V_9$  and  $V_{10}$  in the EMP\_BT relation of Figure 26.9. Here, employee Brown left the company effective August 10, 2002, and the logical delete is carried out by a transaction  $T$  with  $TS(T) = 2002-08-12, 10:11:07$ . Before this,  $V_9$  was the current version of Brown, and its Tet was *uc*. The logical delete is implemented by setting  $V_9.Tet$  to 2002-08-12, 10:11:07 to invalidate it, and creating the *final version*  $V_{10}$  for Brown, with its Vet = 2002-08-10 (see Figure 26.9). Finally, an **insert operation** is implemented by creating the *first version* as illustrated by  $V_{11}$  in the EMP\_BT table.

**Implementation Considerations.** There are various options for storing the tuples in a temporal relation. One is to store all the tuples in the same table, as shown in Figures 26.8 and 26.9. Another option is to create two tables: one for the currently valid information and the other for the rest of the tuples. For example, in the bitemporal EMP\_BT relation, tuples with *uc* for their Tet and *now* for their Vet would be in one relation, the *current table*, since they are the ones currently valid (that is, represent the current snapshot), and all other tuples would be in another relation. This allows the database administrator to have different access paths, such as indexes for each relation, and keeps the size of the current table reasonable. Another possibility is to create a third table for corrected tuples whose Tet is not *uc*.

Another option that is available is to *vertically partition* the attributes of the temporal relation into separate relations so that if a relation has many attributes, a whole new tuple version is created whenever any one of the attributes is updated. If the attributes are updated asynchronously, each new version may differ in only one of the attributes, thus needlessly repeating the other attribute values. If a separate relation is created to contain only the attributes that *always change synchronously*, with the primary key replicated in each relation, the database is said to be in **temporal normal form**. However, to combine the information, a variation of join known as **temporal intersection join** would be needed, which is generally expensive to implement.

It is important to note that bitemporal databases allow a complete record of changes. Even a record of corrections is possible. For example, it is possible that two tuple versions of the same employee may have the same valid time but different attribute values as long as their transaction times are disjoint. In this case, the tuple with the later transaction time is a **correction** of the other tuple version. Even incorrectly entered valid times may be corrected this way. The incorrect state of the database will still be available as a previous database state for querying purposes. A database that keeps such a complete record of changes and corrections is sometimes called an **append-only database**.

### 26.2.3 Incorporating Time in Object-Oriented Databases Using Attribute Versioning

The previous section discussed the **tuple versioning approach** to implementing temporal databases. In this approach, whenever one attribute value is changed, a whole new tuple version is created, even though all the other attribute values will



be identical to the previous tuple version. An alternative approach can be used in database systems that support **complex structured objects**, such as object databases (see Chapter 11) or object-relational systems. This approach is called **attribute versioning**.

In attribute versioning, a single complex object is used to store all the temporal changes of the object. Each attribute that changes over time is called a **time-varying attribute**, and it has its values versioned over time by adding temporal periods to the attribute. The temporal periods may represent valid time, transaction time, or bitemporal, depending on the application requirements. Attributes that do not change over time are called **non-time-varying** and are not associated with the temporal periods. To illustrate this, consider the example in Figure 26.10, which is an attribute-versioned valid time representation of EMPLOYEE

```

class TEMPORAL_SALARY
{
    attribute    Date           Valid_start_time;
    attribute    Date           Valid_end_time;
    attribute    float          Salary;
};

class TEMPORAL_DEPT
{
    attribute    Date           Valid_start_time;
    attribute    Date           Valid_end_time;
    attribute    DEPARTMENT_VT  Dept;
};

class TEMPORAL_SUPERVISOR
{
    attribute    Date           Valid_start_time;
    attribute    Date           Valid_end_time;
    attribute    EMPLOYEE_VT    Supervisor;
};

class TEMPORAL_LIFESPAN
{
    attribute    Date           Valid_ start time;
    attribute    Date           Valid end time;
};

class EMPLOYEE_VT
(
    extent EMPLOYEES )
{
    attribute    list<TEMPORAL_LIFESPAN>    lifespan;
    attribute    string                      Name;
    attribute    string                      Ssn;
    attribute    list<TEMPORAL_SALARY>       Sal_history;
    attribute    list<TEMPORAL_DEPT>         Dept_history;
    attribute    list <TEMPORAL_SUPERVISOR>  Supervisor_history;
};

```

**Figure 26.10**  
Possible ODL schema  
for a temporal valid  
time EMPLOYEE\_VT  
object class using  
attribute versioning.

using the object definition language (ODL) notation for object databases (see Chapter 11). Here, we assumed that name and Social Security number are non-time-varying attributes, whereas salary, department, and supervisor are time-varying attributes (they may change over time). Each time-varying attribute is represented as a list of tuples  $\langle \text{Valid\_start\_time}, \text{Valid\_end\_time}, \text{Value} \rangle$ , ordered by valid start time.

Whenever an attribute is changed in this model, the current attribute version is *closed* and a **new attribute version** for this attribute only is appended to the list. This allows attributes to change asynchronously. The current value for each attribute has *now* for its *Valid\_end\_time*. When using attribute versioning, it is useful to include a **lifespan temporal attribute** associated with the whole object whose value is one or more valid time periods that indicate the valid time of existence for the whole object. Logical deletion of the object is implemented by closing the lifespan. The constraint that any time period of an attribute within an object should be a subset of the object's lifespan should be enforced.

For bitemporal databases, each attribute version would have a tuple with five components:

$\langle \text{Valid\_start\_time}, \text{Valid\_end\_time}, \text{Trans\_start\_time}, \text{Trans\_end\_time}, \text{Value} \rangle$

The object lifespan would also include both valid and transaction time dimensions. Therefore, the full capabilities of bitemporal databases can be available with attribute versioning. Mechanisms similar to those discussed earlier for updating tuple versions can be applied to updating attribute versions.

## 26.2.4 Temporal Querying Constructs and the TSQL2 Language

So far, we have discussed how data models may be extended with temporal constructs. Now we give a brief overview of how query operations need to be extended for temporal querying. We will briefly discuss the TSQL2 language, which extends SQL for querying valid time and transaction time tables, as well as for querying of bitemporal relational tables.

In nontemporal relational databases, the typical selection conditions involve attribute conditions, and tuples that satisfy these conditions are selected from the set of *current tuples*. Following that, the attributes of interest to the query are specified by a *projection operation* (see Chapter 6). For example, in the query to retrieve the names of all employees working in department 5 whose salary is greater than 30000, the selection condition would be as follows:

$((\text{Salary} > 30000) \text{ AND } (\text{Dno} = 5))$

The projected attribute would be Name. In a temporal database, the conditions may involve time in addition to attributes. A **pure time condition** involves only time—for example, to select all employee tuple versions that were valid on a certain *time point*  $T$  or that were valid *during a certain time period*  $[T_1, T_2]$ . In this

case, the specified time period is compared with the valid time period of each tuple version  $[T.Vst, T.Vet]$ , and only those tuples that satisfy the condition are selected. In these operations, a period is considered to be equivalent to the set of time points from  $T_1$  to  $T_2$  inclusive, so the standard set comparison operations can be used. Additional operations, such as whether one time period ends *before* another starts, are also needed.<sup>21</sup>

Some of the more common operations used in queries are as follows:

$[T.Vst, T.Vet]$ <b>INCLUDES</b> $[T_1, T_2]$	Equivalent to $T_1 \geq T.Vst$ AND $T_2 \leq T.Vet$
$[T.Vst, T.Vet]$ <b>INCLUDED_IN</b> $[T_1, T_2]$	Equivalent to $T_1 \leq T.Vst$ AND $T_2 \geq T.Vet$
$[T.Vst, T.Vet]$ <b>OVERLAPS</b> $[T_1, T_2]$	Equivalent to $(T_1 \leq T.Vet$ AND $T_2 \geq T.Vst)$ <sup>22</sup>
$[T.Vst, T.Vet]$ <b>BEFORE</b> $[T_1, T_2]$	Equivalent to $T_1 \geq T.Vet$
$[T.Vst, T.Vet]$ <b>AFTER</b> $[T_1, T_2]$	Equivalent to $T_2 \leq T.Vst$
$[T.Vst, T.Vet]$ <b>MEETS_BEFORE</b> $[T_1, T_2]$	Equivalent to $T_1 = T.Vet + 1$ <sup>23</sup>
$[T.Vst, T.Vet]$ <b>MEETS_AFTER</b> $[T_1, T_2]$	Equivalent to $T_2 + 1 = T.Vst$

Additionally, operations are needed to manipulate time periods, such as computing the union or intersection of two time periods. The results of these operations may not themselves be periods, but rather **temporal elements**—a collection of one or more *disjoint* time periods such that no two time periods in a temporal element are directly adjacent. That is, for any two time periods  $[T_1, T_2]$  and  $[T_3, T_4]$  in a temporal element, the following three conditions must hold:

- $[T_1, T_2]$  intersection  $[T_3, T_4]$  is empty.
- $T_3$  is not the time point following  $T_2$  in the given granularity.
- $T_1$  is not the time point following  $T_4$  in the given granularity.

The latter conditions are necessary to ensure unique representations of temporal elements. If two time periods  $[T_1, T_2]$  and  $[T_3, T_4]$  are adjacent, they are combined into a single time period  $[T_1, T_4]$ . This is called **coalescing** of time periods. Coalescing also combines intersecting time periods.

To illustrate how pure time conditions can be used, suppose a user wants to select all employee versions that were valid at any point during 2002. The appropriate selection condition applied to the relation in Figure 26.8 would be

$[T.Vst, T.Vet]$  **OVERLAPS** [2002-01-01, 2002-12-31]

Typically, most temporal selections are applied to the valid time dimension. For a bitemporal database, one usually applies the conditions to the currently correct

<sup>21</sup>A complete set of operations, known as **Allen's algebra** (Allen, 1983), has been defined for comparing time periods.

<sup>22</sup>This operation returns true if the *intersection* of the two periods is not empty; it has also been called INTERSECTS\_WITH.

<sup>23</sup>Here, 1 refers to one time point in the specified granularity. The MEETS operations basically specify if one period starts immediately after another period ends.

tuples with *uc* as their transaction end times. However, if the query needs to be applied to a previous database state, an *AS\_OF T* clause is appended to the query, which means that the query is applied to the valid time tuples that were correct in the database at time *T*.

In addition to pure time conditions, other selections involve **attribute and time conditions**. For example, suppose we wish to retrieve all EMP\_VT tuple versions *T* for employees who worked in department 5 at any time during 2002. In this case, the condition is

[*T.Vst*, *T.Vet*]**OVERLAPS** [2002-01-01, 2002-12-31] AND (*T.Dno* = 5)

Finally, we give a brief overview of the TSQL2 query language, which extends SQL with constructs for temporal databases. The main idea behind TSQL2 is to allow users to specify whether a relation is nontemporal (that is, a standard SQL relation) or temporal. The CREATE TABLE statement is extended with an *optional* AS clause to allow users to declare different temporal options. The following options are available:

- AS VALID STATE <GRANULARITY> (valid time relation with valid time period)
- AS VALID EVENT <GRANULARITY> (valid time relation with valid time point)
- AS TRANSACTION (transaction time relation with transaction time period)
- AS VALID STATE <GRANULARITY> AND TRANSACTION (bitemporal relation, valid time period)
- AS VALID EVENT <GRANULARITY> AND TRANSACTION (bitemporal relation, valid time point)

The keywords STATE and EVENT are used to specify whether a time *period* or time *point* is associated with the valid time dimension. In TSQL2, rather than have the user actually see how the temporal tables are implemented (as we discussed in the previous sections), the TSQL2 language adds query language constructs to specify various types of temporal selections, temporal projections, temporal aggregations, transformation among granularities, and many other concepts. The book by Snodgrass et al. (1995) describes the language.

### 26.2.5 Time Series Data

Time series data is used very often in financial, sales, and economics applications. They involve data values that are recorded according to a specific predefined sequence of time points. Therefore, they are a special type of **valid event data**, where the event's time points are predetermined according to a fixed calendar. Consider the example of closing daily stock prices of a particular company on the New York Stock Exchange. The granularity here is day, but the days that the stock market is open are known (non-holiday weekdays). Hence, it has been common to specify a computational procedure that calculates the particular **calendar** associated with a time series. Typical queries on

time series involve **temporal aggregation** over higher granularity intervals—for example, finding the average or maximum *weekly* closing stock price or the maximum and minimum *monthly* closing stock price from the *daily* information.

As another example, consider the daily sales dollar amount at each store of a chain of stores owned by a particular company. Again, typical temporal aggregates would be retrieving the weekly, monthly, or yearly sales from the daily sales information (using the sum aggregate function), or comparing same store monthly sales with previous monthly sales, and so on.

Because of the specialized nature of time series data and the lack of support for it in older DBMSs, it has been common to use specialized **time series management systems** rather than general-purpose DBMSs for managing such information. In such systems, it has been common to store time series values in sequential order in a file and apply specialized time series procedures to analyze the information. The problem with this approach is that the full power of high-level querying in languages such as SQL will not be available in such systems.

More recently, some commercial DBMS packages began offering time series extensions, such as the Oracle time cartridge and the time series data blade of Informix Universal Server. In addition, the TSQL2 language provides some support for time series in the form of event tables.

## 26.3 Spatial Database Concepts<sup>24</sup>

### 26.3.1 Introduction to Spatial Databases

Spatial databases incorporate functionality that provides support for databases that keep track of objects in a multidimensional space. For example, cartographic databases that store maps include two-dimensional spatial descriptions of their objects—from countries and states to rivers, cities, roads, seas, and so on. The systems that manage geographic data and related applications are known as **geographic information systems (GISs)**, and they are used in areas such as environmental applications, transportation systems, emergency response systems, and battle management. Other databases, such as meteorological databases for weather information, are three-dimensional, since temperatures and other meteorological information are related to three-dimensional spatial points. In general, a **spatial database** stores objects that have spatial characteristics that describe them and that have spatial relationships among them. The spatial relationships among the objects are important, and they are often needed when querying the database. Although a spatial database can in general refer to an  $n$ -dimensional space for any  $n$ , we will limit our discussion to two dimensions as an illustration.

A spatial database is optimized to store and query data related to objects in space, including points, lines and polygons. Satellite images are a prominent example of

---

<sup>24</sup>The contribution of Pranesh Parimala Ranganathan to this section is appreciated.

spatial data. Queries posed on these spatial data, where predicates for selection deal with spatial parameters, are called **spatial queries**. For example, “What are the names of all bookstores within five miles of the College of Computing building at Georgia Tech?” is a spatial query. Whereas typical databases process numeric and character data, additional functionality needs to be added for databases to process spatial data types. A query such as “List all the customers located within twenty miles of company headquarters” will require the processing of spatial data types typically outside the scope of standard relational algebra and may involve consulting an external geographic database that maps the company headquarters and each customer to a 2-D map based on their address. Effectively, each customer will be associated to a <latitude, longitude> position. A traditional B<sup>+</sup>-tree index based on customers’ zip codes or other nonspatial attributes cannot be used to process this query since traditional indexes are not capable of ordering multidimensional coordinate data. Therefore, there is a special need for databases tailored for handling spatial data and spatial queries.

Table 26.1 shows the common analytical operations involved in processing geographic or spatial data.<sup>25</sup> **Measurement operations** are used to measure some global properties of single objects (such as the area, the relative size of an object’s parts, compactness, or symmetry) and to measure the relative position of different objects in terms of distance and direction. **Spatial analysis** operations, which often use statistical techniques, are used to uncover *spatial relationships* within and among mapped data layers. An example would be to create a map—known as a *prediction map*—that identifies the locations of likely customers for particular products based on the historical sales and demographic information. **Flow analysis** operations help in determining the shortest path between two points and also the connectivity among nodes or regions in a graph. **Location analysis** aims to find if the given set of points and lines lie within a given polygon (location). The process involves generating a buffer around existing geographic features and then identifying or selecting features based on whether they fall inside or outside the boundary of the buffer. **Digital terrain analysis** is used to build three-dimensional models,

**Table 26.1** Common Types of Analysis for Spatial Data

Analysis Type	Type of Operations and Measurements
Measurements	Distance, perimeter, shape, adjacency, and direction
Spatial analysis/statistics	Pattern, autocorrelation, and indexes of similarity and topology using spatial and nonspatial data
Flow analysis	Connectivity and shortest path
Location analysis	Analysis of points and lines within a polygon
Terrain analysis	Slope/aspect, catchment area, drainage network
Search	Thematic search, search by region

<sup>25</sup>List of GIS analysis operations as proposed in Albrecht (1996).

where the topography of a geographical location can be represented with an  $x, y, z$  data model known as Digital Terrain (or Elevation) Model (DTM/DEM). The  $x$  and  $y$  dimensions of a DTM represent the horizontal plane, and  $z$  represents spot heights for the respective  $x, y$  coordinates. Such models can be used for analysis of environmental data or during the design of engineering projects that require terrain information. Spatial search allows a user to search for objects within a particular spatial region. For example, **thematic search** allows us to search for objects related to a particular theme or class, such as “Find all water bodies within 25 miles of Atlanta” where the class is *water*.

There are also **topological relationships** among spatial objects. These are often used in Boolean predicates to select objects based on their spatial relationships. For example, if a city boundary is represented as a polygon and freeways are represented as multilines, a condition such as “Find all freeways that go through Arlington, Texas” would involve an *intersects* operation, to determine which freeways (lines) intersect the city boundary (polygon).

### 26.3.2 Spatial Data Types and Models

This section briefly describes the common data types and models for storing spatial data. Spatial data comes in three basic forms. These forms have become a *de facto* standard due to their wide use in commercial systems.

- **Map data**<sup>26</sup> includes various geographic or spatial features of objects in a map, such as an object’s shape and the location of the object within the map. The three basic types of features are points, lines, and polygons (or areas). **Points** are used to represent spatial characteristics of objects whose locations correspond to a single 2-D coordinate ( $x, y$ , or longitude/latitude) in the scale of a particular application. Depending on the scale, some examples of point objects could be buildings, cellular towers, or stationary vehicles. Moving vehicles and other moving objects can be represented by a sequence of point locations that change over time. **Lines** represent objects having length, such as roads or rivers, whose spatial characteristics can be approximated by a sequence of connected lines. **Polygons** are used to represent spatial characteristics of objects that have a boundary, such as countries, states, lakes, or cities. Notice that some objects, such as buildings or cities, can be represented as either points or polygons, depending on the scale of detail.
- **Attribute data** is the descriptive data that GIS systems associate with **map features**. For example, suppose that a map contains features that represent counties within a U.S. state (such as Texas or Oregon). Attributes for each county feature (object) could include population, largest city/town, area in square miles, and so on. Other attribute data could be included for other features in the map, such as states, cities, congressional districts, census tracts, and so on.

---

<sup>26</sup>These types of geographic data are based on ESRI’s guide to GIS. See [www.gis.com/implementing\\_gis/data/data\\_types.html](http://www.gis.com/implementing_gis/data/data_types.html)



- **Image data** includes data such as satellite images and aerial photographs, which are typically created by cameras. Objects of interest, such as buildings and roads, can be identified and overlaid on these images. Images can also be attributes of map features. One can add images to other map features so that clicking on the feature would display the image. Aerial and satellite images are typical examples of raster data.

**Models of spatial information** are sometimes grouped into two broad categories: *field* and *object*. A spatial application (such as remote sensing or highway traffic control) is modeled using either a field- or an object-based model, depending on the requirements and the traditional choice of model for the application. **Field models** are often used to model spatial data that is continuous in nature, such as terrain elevation, temperature data, and soil variation characteristics, whereas **object models** have traditionally been used for applications such as transportation networks, land parcels, buildings, and other objects that possess both spatial and non-spatial attributes.

### 26.3.3 Spatial Operators and Spatial Queries

Spatial operators are used to capture all the relevant geometric properties of objects embedded in the physical space and the relations between them, as well as to perform spatial analysis. Operators are classified into three broad categories.

- **Topological operators.** Topological properties are invariant when topological transformations are applied. These properties do not change after transformations like rotation, translation, or scaling. Topological operators are hierarchically structured in several levels, where the base level offers operators the ability to check for detailed topological relations between regions with a broad boundary, and the higher levels offer more abstract operators that allow users to query uncertain spatial data independent of the underlying geometric data model. Examples include open (region), close (region), and inside (point, loop).
- **Projective operators.** Projective operators, such as *convex hull*, are used to express predicates about the concavity/convexity of objects as well as other spatial relations (for example, being inside the concavity of a given object).
- **Metric operators.** Metric operators provide a more specific description of the object's geometry. They are used to measure some global properties of single objects (such as the area, relative size of an object's parts, compactness, and symmetry), and to measure the relative position of different objects in terms of distance and direction. Examples include length (arc) and distance (point, point).

**Dynamic Spatial Operators.** The operations performed by the operators mentioned above are static, in the sense that the operands are not affected by the application of the operation. For example, calculating the length of the curve has no effect on the curve itself. **Dynamic operations** alter the objects upon which the operations act. The three fundamental dynamic operations are *create*, *destroy*, and



*update*. A representative example of dynamic operations would be updating a spatial object that can be subdivided into translate (shift position), rotate (change orientation), scale up or down, reflect (produce a mirror image), and shear (deform).

**Spatial Queries.** Spatial queries are requests for spatial data that require the use of spatial operations. The following categories illustrate three typical types of spatial queries:

- **Range queries.** Find all objects of a particular type that are within a given spatial area; for example, find all hospitals within the Metropolitan Atlanta city area. A variation of this query is to find all objects within a particular distance from a given location; for example, find all ambulances within a five mile radius of an accident location.
- **Nearest neighbor queries.** Finds an object of a particular type that is closest to a given location; for example, find the police car that is closest to the location of a crime. This can be generalized to find the  $k$  nearest neighbors, such as the 5 closest ambulances to an accident location.
- **Spatial joins or overlays.** Typically joins the objects of two types based on some spatial condition, such as the objects intersecting or overlapping spatially or being within a certain distance of one another. For example, find all townships located on a major highway between two cities or find all homes that are within two miles of a lake. The first example spatially joins *township* objects and *highway* object, and the second example spatially joins *lake* objects and *home* objects.

### 26.3.4 Spatial Data Indexing

A spatial index is used to organize objects into a set of buckets (which correspond to pages of secondary memory), so that objects in a particular spatial region can be easily located. Each bucket has a bucket region, a part of space containing all objects stored in the bucket. The bucket regions are usually rectangles; for point data structures, these regions are disjoint and they partition the space so that each point belongs to precisely one bucket. There are essentially two

1. Specialized indexing structures that allow efficient search for data objects based on spatial search operations are included in the database system. These indexing structures would play a similar role to that performed by  $B^+$ -tree indexes in traditional database systems. Examples of these indexing structures are *grid files* and *R-trees*. Special types of spatial indexes, known as *spatial join indexes*, can be used to speed up spatial join operations.
2. Instead of creating brand new indexing structures, the two-dimensional (2-D) spatial data is converted to single-dimensional (1-D) data, so that traditional indexing techniques ( $B^+$ -tree) can be used. The algorithms for converting from 2-D to 1-D are known as *space filling curves*. We will not discuss these methods in detail (see the Selected Bibliography for further references).

We give an overview of some of the spatial indexing techniques next.

**Grid Files.** We introduced grid files for indexing of data on multiple attributes in Chapter 18. They can also be used for indexing two-dimensional and higher  $n$ -dimensional spatial data. The **fixed-grid** method divides an  $n$ -dimensional hyperspace into equal size buckets. The data structure that implements the fixed grid is an  $n$ -dimensional array. The objects whose spatial locations lie within a cell (totally or partially) can be stored in a dynamic structure to handle overflows. This structure is useful for uniformly distributed data like satellite imagery. However, the fixed-grid structure is rigid, and its directory can be sparse and large.

**R-Trees.** The **R-tree** is a height-balanced tree, which is an extension of the  $B^+$ -tree for  $k$ -dimensions, where  $k > 1$ . For two dimensions (2-D), spatial objects are approximated in the  $R$ -tree by their **minimum bounding rectangle (MBR)**, which is the smallest rectangle, with sides parallel to the coordinate system ( $x$  and  $y$ ) axis, that contains the object.  $R$ -trees are characterized by the following properties, which are similar to the properties for  $B^+$ -trees (see Section 18.3) but are adapted to 2-D spatial objects. As in Section 18.3, we use  $M$  to indicate the maximum number of entries that can fit in an  $R$ -tree node.

1. The structure of each index entry (or index record) in a leaf node is  $(I, \text{object-identifier})$ , where  $I$  is the MBR for the spatial object whose identifier is *object-identifier*.
2. Every node except the root node must be at least half full. Thus, a leaf node that is not the root should contain  $m$  entries  $(I, \text{object-identifier})$  where  $M/2 \leq m \leq M$ . Similarly, a non-leaf node that is not the root should contain  $m$  entries  $(I, \text{child-pointer})$  where  $M/2 \leq m \leq M$ , and  $I$  is the MBR that contains the union of all the rectangles in the node pointed at by *child-pointer*.
3. All leaf nodes are at the same level, and the root node should have at least two pointers unless it is a leaf node.
4. All MBRs have their sides parallel to the axes of the global coordinate system.

Other spatial storage structures include quadtrees and their variations. **Quadtrees** generally divide each space or subspace into equally sized areas and proceed with the subdivisions of each subspace to identify the positions of various objects. Recently, many newer spatial access structures have been proposed, and this remains an active research area.

**Spatial Join Index.** A spatial join index precomputes a spatial join operation and stores the pointers to the related object in an index structure. Join indexes improve the performance of recurring join queries over tables that have low update rates. Spatial join conditions are used to answer queries such as “Create a list of highway-river combinations that cross.” The spatial join is used to identify and retrieve these pairs of objects that satisfy the *cross* spatial relationship. Because computing the results of spatial relationships is generally time consuming, the result can be computed once and stored in a table that has the pairs of object identifiers (or tuple ids) that satisfy the spatial relationship, which is essentially the join index.

A join index can be described by a bipartite graph  $G = (V_1, V_2, E)$ , where  $V_1$  contains the tuple ids of relation  $R$  and  $V_2$  contains the tuple ids of relation  $S$ . Edge set contains an edge  $(v_r, v_s)$  for  $v_r$  in  $R$  and  $v_s$  in  $S$ , if there is a tuple corresponding to  $(v_r, v_s)$  in the join index. The bipartite graph models all of the related tuples as connected vertices in the graphs. Spatial join indexes are used in operations (see Section 26.3.3) that involve computation of relationships among spatial objects.

### 26.3.5 Spatial Data Mining

Spatial data tends to be highly correlated. For example, people with similar characteristics, occupations, and backgrounds tend to cluster together in the same neighborhoods. The three major spatial data mining techniques are spatial classification, spatial association, and spatial clustering.

- **Spatial classification.** The goal of classification is to estimate the value of an attribute of a relation based on the value of the relation's other attributes. An example of the spatial classification problem is determining the locations of nests in a wetland based on the value of other attributes (for example, vegetation durability and water depth); it is also called the *location prediction problem*. Similarly, where to expect hotspots in crime activity is also a location prediction problem.
- **Spatial association. Spatial association rules** are defined in terms of spatial predicates rather than items. A spatial association rule is of the form

$$P_1 \wedge P_2 \wedge \dots \wedge P_n \Rightarrow Q_1 \wedge Q_2 \wedge \dots \wedge Q_m$$

where at least one of the  $P_i$ 's or  $Q_j$ 's is a spatial predicate. For example, the rule

$$\text{is\_a}(x, \text{country}) \wedge \text{touches}(x, \text{Mediterranean}) \Rightarrow \text{is\_a}(x, \text{wine-exporter})$$

(that is, a country that is adjacent to the Mediterranean Sea is typically a wine exporter) is an example of an association rule, which will have a certain support  $s$  and confidence  $c$ .<sup>27</sup>

**Spatial colocation rules** attempt to generalize association rules to point to collection data sets that are indexed by space. There are several crucial differences between spatial and nonspatial associations, including the following:

1. The notion of a transaction is absent in spatial situations, since data is embedded in continuous space. Partitioning space into transactions would lead to an overestimate or an underestimate of interest measures, for example, support or confidence.
2. Size of item sets in spatial databases is small, that is, there are many fewer items in the item set in a spatial situation than in a nonspatial situation.

<sup>27</sup>Concepts of support and confidence for association rules are discussed as part of data mining in Section 28.2.

In most instances, spatial items are a discrete version of continuous variables. For example, in the United States income regions may be defined as regions where the mean yearly income is within certain ranges, such as, below \$40,000, from \$40,000 to \$100,000, and above \$100,000.

- **Spatial clustering** attempts to group database objects so that the most similar objects are in the same cluster, and objects in different clusters are as dissimilar as possible. One application of spatial clustering is to group together seismic events in order to determine earthquake faults. An example of a spatial clustering algorithm is **density-based clustering**, which tries to find clusters based on the density of data points in a region. These algorithms treat clusters as dense regions of objects in the data space. Two variations of these algorithms are density-based spatial clustering of applications with noise (DBSCAN)<sup>28</sup> and density-based clustering (DENCLUE).<sup>29</sup> DBSCAN is a density-based clustering algorithm because it finds a number of clusters starting from the estimated density distribution of corresponding nodes.

### 26.3.6 Applications of Spatial Data

Spatial data management is useful in many disciplines, including geography, remote sensing, urban planning, and natural resource management. Spatial database management is playing an important role in the solution of challenging scientific problems such as global climate change and genomics. Due to the spatial nature of genome data, GIS and spatial database management systems have a large role to play in the area of bioinformatics. Some of the typical applications include pattern recognition (for example, to check if the topology of a particular gene in the genome is found in any other sequence feature map in the database), genome browser development, and visualization maps. Another important application area of spatial data mining is the spatial outlier detection. A **spatial outlier** is a spatially referenced object whose nonspatial attribute values are significantly different from those of other spatially referenced objects in its spatial neighborhood. For example, if a neighborhood of older houses has just one brand-new house, that house would be an outlier based on the nonspatial attribute 'house\_age'. Detecting spatial outliers is useful in many applications of geographic information systems and spatial databases. These application domains include transportation, ecology, public safety, public health, climatology, and location-based services.

## 26.4 Multimedia Database Concepts

**Multimedia databases** provide features that allow users to store and query different types of multimedia information, which includes *images* (such as photos or drawings), *video clips* (such as movies, newsreels, or home videos), *audio clips*

<sup>28</sup>DBSCAN was proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu (1996).

<sup>29</sup>DENCLUE was proposed by Hinnenberg and Gabriel (2007).

(such as songs, phone messages, or speeches), and *documents* (such as books or articles). The main types of database queries that are needed involve locating multimedia sources that contain certain objects of interest. For example, one may want to locate all video clips in a video database that include a certain person, say Michael Jackson. One may also want to retrieve video clips based on certain activities included in them, such as video clips where a soccer goal is scored by a certain player or team.

The above types of queries are referred to as **content-based retrieval**, because the multimedia source is being retrieved based on its containing certain objects or activities. Hence, a multimedia database must use some model to organize and index the multimedia sources based on their contents. *Identifying the contents* of multimedia sources is a difficult and time-consuming task. There are two main approaches. The first is based on **automatic analysis** of the multimedia sources to identify certain mathematical characteristics of their contents. This approach uses different techniques depending on the type of multimedia source (image, video, audio, or text). The second approach depends on **manual identification** of the objects and activities of interest in each multimedia source and on using this information to index the sources. This approach can be applied to all multimedia sources, but it requires a manual preprocessing phase in which a person must scan each multimedia source to identify and catalog the objects and activities it contains so that they can be used to index the sources.

In the first part of this section, we will briefly discuss some of the characteristics of each type of multimedia source—images, video, audio, and text/documents. Then we will discuss approaches for automatic analysis of images followed by the problem of object recognition in images. We end this section with some remarks on analyzing audio sources.

An **image** is typically stored either in raw form as a set of pixel or cell values, or in compressed form to save space. The image *shape descriptor* describes the geometric shape of the raw image, which is typically a rectangle of **cells** of a certain width and height. Hence, each image can be represented by an  $m$  by  $n$  grid of cells. Each cell contains a pixel value that describes the cell content. In black-and-white images, pixels can be one bit. In grayscale or color images, a pixel is multiple bits. Because images may require large amounts of space, they are often stored in compressed form. Compression standards, such as GIF, JPEG, or MPEG, use various mathematical transformations to reduce the number of cells stored but still maintain the main image characteristics. Applicable mathematical transforms include discrete Fourier transform (DFT), discrete cosine transform (DCT), and wavelet transforms.

To identify objects of interest in an image, the image is typically divided into homogeneous segments using a *homogeneity predicate*. For example, in a color image, adjacent cells that have similar pixel values are grouped into a segment. The homogeneity predicate defines conditions for automatically grouping those cells. Segmentation and compression can hence identify the main characteristics of an image.

A typical image database query would be to find images in the database that are similar to a given image. The given image could be an isolated segment that contains, say, a pattern of interest, and the query is to locate other images that contain that same pattern. There are two main techniques for this type of search. The first approach uses a **distance function** to compare the given image with the stored images and their segments. If the distance value returned is small, the probability of a match is high. Indexes can be created to group stored images that are close in the distance metric so as to limit the search space. The second approach, called the **transformation approach**, measures image similarity by having a small number of transformations that can change one image's cells to match the other image. Transformations include rotations, translations, and scaling. Although the transformation approach is more general, it is also more time-consuming and difficult.

A **video source** is typically represented as a sequence of frames, where each frame is a still image. However, rather than identifying the objects and activities in every individual frame, the video is divided into **video segments**, where each segment comprises a sequence of contiguous frames that includes the same objects/activities. Each segment is identified by its starting and ending frames. The objects and activities identified in each video segment can be used to index the segments. An indexing technique called *frame segment trees* has been proposed for video indexing. The index includes both objects, such as persons, houses, and cars, as well as activities, such as a person *delivering* a speech or two people *talking*. Videos are also often compressed using standards such as MPEG.

**Audio sources** include stored recorded messages, such as speeches, class presentations, or even surveillance recordings of phone messages or conversations by law enforcement. Here, discrete transforms can be used to identify the main characteristics of a certain person's voice in order to have similarity-based indexing and retrieval. We will briefly comment on their analysis in Section 26.4.4.

A **text/document source** is basically the full text of some article, book, or magazine. These sources are typically indexed by identifying the keywords that appear in the text and their relative frequencies. However, filler words or common words called **stopwords** are eliminated from the process. Because there can be many keywords when attempting to index a collection of documents, techniques have been developed to reduce the number of keywords to those that are most relevant to the collection. A dimensionality reduction technique called *singular value decomposition* (SVD), which is based on matrix transformations, can be used for this purpose. An indexing technique called *telescoping vector trees* (TV-trees) can then be used to group similar documents. Chapter 27 discusses document processing in detail.

### 26.4.1 Automatic Analysis of Images

Analysis of multimedia sources is critical to support any type of query or search interface. We need to represent multimedia source data such as images in terms of features that would enable us to define similarity. The work done so far in this area uses low-level visual features such as color, texture, and shape, which are directly

related to the perceptual aspects of image content. These features are easy to extract and represent, and it is convenient to design similarity measures based on their statistical properties.

**Color** is one of the most widely used visual features in content-based image retrieval since it does not depend upon image size or orientation. Retrieval based on color similarity is mainly done by computing a color histogram for each image that identifies the proportion of pixels within an image for the three color channels (red, green, blue—**RGB**). However, RGB representation is affected by the orientation of the object with respect to illumination and camera direction. Therefore, current image retrieval techniques compute color histograms using competing invariant representations such as **HSV** (hue, saturation, value). HSV describes colors as points in a cylinder whose central axis ranges from black at the bottom to white at the top with neutral colors between them. The angle around the axis corresponds to the hue, the distance from the axis corresponds to the saturation, and the distance along the axis corresponds to the value (brightness).

**Texture** refers to the patterns in an image that present the properties of homogeneity that do not result from the presence of a single color or intensity value. Examples of texture classes are rough and silky. Examples of textures that can be identified include pressed calf leather, straw matting, cotton canvas, and so on. Just as pictures are represented by arrays of pixels (picture elements), textures are represented by **arrays of texels** (texture elements). These textures are then placed into a number of sets, depending on how many textures are identified in the image. These sets not only contain the texture definition but also indicate where in the image the texture is located. Texture identification is primarily done by modeling it as a two-dimensional, gray-level variation. The relative brightness of pairs of pixels is computed to estimate the degree of contrast, regularity, coarseness, and directionality.

**Shape** refers to the shape of a region within an image. It is generally determined by applying segmentation or edge detection to an image. **Segmentation** is a region-based approach that uses an entire region (sets of pixels), whereas **edge detection** is a boundary-based approach that uses only the outer boundary characteristics of entities. Shape representation is typically required to be invariant to translation, rotation, and scaling. Some well-known methods for shape representation include Fourier descriptors and moment invariants.

## 26.4.2 Object Recognition in Images

**Object recognition** is the task of identifying real-world objects in an image or a video sequence. The system must be able to identify the object even when the images of the object vary in viewpoints, size, scale, or even when they are rotated or translated. Some approaches have been developed to divide the original image into regions based on similarity of contiguous pixels. Thus, in a given image showing a tiger in the jungle, a tiger subimage may be detected against the background of the jungle, and when compared with a set of training images, it may be tagged as a tiger.



The representation of the multimedia object in an object model is extremely important. One approach is to divide the image into homogeneous segments using a homogeneous predicate. For example, in a colored image, adjacent cells that have similar pixel values are grouped into a segment. The homogeneity predicate defines conditions for automatically grouping those cells. Segmentation and compression can hence identify the main characteristics of an image. Another approach finds measurements of the object that are invariant to transformations. It is impossible to keep a database of examples of all the different transformations of an image. To deal with this, object recognition approaches find interesting points (or features) in an image that are invariant to transformations.

An important contribution to this field was made by Lowe,<sup>30</sup> who used scale-invariant features from images to perform reliable object recognition. This approach is called **scale-invariant feature transform (SIFT)**. The SIFT features are invariant to image scaling and rotation, and partially invariant to change in illumination and 3D camera viewpoint. They are well localized in both the spatial and frequency domains, reducing the probability of disruption by occlusion, clutter, or noise. In addition, the features are highly distinctive, which allows a single feature to be correctly matched with high probability against a large database of features, providing a basis for object and scene recognition.

For image matching and recognition, SIFT features (also known as *keypoint features*) are first extracted from a set of reference images and stored in a database. Object recognition is then performed by comparing each feature from the new image with the features stored in the database and finding candidate matching features based on the Euclidean distance of their feature vectors. Since the keypoint features are highly distinctive, a single feature can be correctly matched with good probability in a large database of features.

In addition to SIFT, there are a number of competing methods available for object recognition under clutter or partial occlusion. For example, **RIFT**, a rotation invariant generalization of SIFT, identifies groups of local affine regions (image features having a characteristic appearance and elliptical shape) that remain approximately affinely rigid across a range of views of an object, and across multiple instances of the same object class.

### 26.4.3 Semantic Tagging of Images

The notion of implicit tagging is an important one for image recognition and comparison. Multiple tags may attach to an image or a subimage: for instance, in the example we referred to above, tags such as “tiger,” “jungle,” “green,” and “stripes” may be associated with that image. Most image search techniques retrieve images based on user-supplied tags that are often not very accurate or comprehensive. To improve search quality, a number of recent systems aim at automated generation of these image tags. In case of multimedia data, most of its semantics is present in its

---

<sup>30</sup>See Lowe (2004), “Distinctive Image Features from Scale-Invariant Keypoints.”



content. These systems use image-processing and statistical-modeling techniques to analyze image content to generate accurate annotation tags that can then be used to retrieve images by content. Since different annotation schemes will use different vocabularies to annotate images, the quality of image retrieval will be poor. To solve this problem, recent research techniques have proposed the use of concept hierarchies, taxonomies, or ontologies using **OWL (Web Ontology Language)**, in which terms and their relationships are clearly defined. These can be used to infer higher-level concepts based on tags. Concepts like “sky” and “grass” may be further divided into “clear sky” and “cloudy sky” or “dry grass” and “green grass” in such a taxonomy. These approaches generally come under semantic tagging and can be used in conjunction with the above feature-analysis and object-identification strategies.

#### 26.4.4 Analysis of Audio Data Sources

Audio sources are broadly classified into speech, music, and other audio data. Each of these is significantly different from the others; hence different types of audio data are treated differently. Audio data must be digitized before it can be processed and stored. Indexing and retrieval of audio data is arguably the toughest among all types of media, because like video, it is continuous in time and does not have easily measurable characteristics such as text. Clarity of sound recordings is easy to perceive humanly but is hard to quantify for machine learning. Interestingly, speech data often uses speech recognition techniques to aid the actual audio content, as this can make indexing this data a lot easier and more accurate. This is sometimes referred to as *text-based indexing of audio data*. The speech metadata is typically content dependent, in that the metadata is generated from the audio content; for example, the length of the speech, the number of speakers, and so on. However, some of the metadata might be independent of the actual content, such as the length of the speech and the format in which the data is stored. Music indexing, on the other hand, is done based on the statistical analysis of the audio signal, also known as *content-based indexing*. Content-based indexing often makes use of the key features of sound: intensity, pitch, timbre, and rhythm. It is possible to compare different pieces of audio data and retrieve information from them based on the calculation of certain features, as well as application of certain transforms.

## 26.5 Introduction to Deductive Databases

### 26.5.1 Overview of Deductive Databases

In a deductive database system we typically specify rules through a **declarative language**—a language in which we specify what to achieve rather than how to achieve it. An **inference engine** (or **deduction mechanism**) within the system can deduce new facts from the database by interpreting these rules. The model used for deductive databases is closely related to the relational data model, and particularly to the domain relational calculus formalism (see Section 6.6). It is also related to the field of **logic programming** and the **Prolog** language. The deductive database work

based on logic has used Prolog as a starting point. A variation of Prolog called **Datalog** is used to define rules declaratively in conjunction with an existing set of relations, which are themselves treated as literals in the language. Although the language structure of Datalog resembles that of Prolog, its operational semantics—that is, how a Datalog program is executed—is still different.

A deductive database uses two main types of specifications: facts and rules. **Facts** are specified in a manner similar to the way relations are specified, except that it is not necessary to include the attribute names. Recall that a tuple in a relation describes some real-world fact whose meaning is partly determined by the attribute names. In a deductive database, the meaning of an attribute value in a tuple is determined solely by its *position* within the tuple. **Rules** are somewhat similar to relational views. They specify virtual relations that are not actually stored but that can be formed from the facts by applying inference mechanisms based on the rule specifications. The main difference between rules and views is that rules may involve recursion and hence may yield virtual relations that cannot be defined in terms of basic relational views.

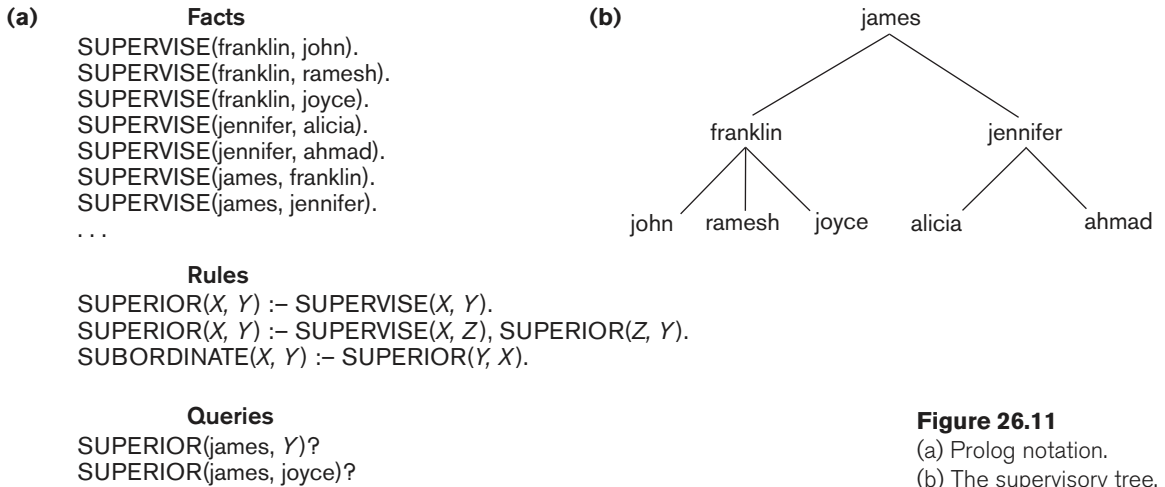
The evaluation of Prolog programs is based on a technique called *backward chaining*, which involves a top-down evaluation of goals. In the deductive databases that use Datalog, attention has been devoted to handling large volumes of data stored in a relational database. Hence, evaluation techniques have been devised that resemble those for a bottom-up evaluation. Prolog suffers from the limitation that the order of specification of facts and rules is significant in evaluation; moreover, the order of literals (defined in Section 26.5.3) within a rule is significant. The execution techniques for Datalog programs attempt to circumvent these problems.

## 26.5.2 Prolog/Datalog Notation

The notation used in Prolog/Datalog is based on providing predicates with unique names. A **predicate** has an implicit meaning, which is suggested by the predicate name, and a fixed number of **arguments**. If the arguments are all constant values, the predicate simply states that a certain fact is true. If, on the other hand, the predicate has variables as arguments, it is either considered as a query or as part of a rule or constraint. In our discussion, we adopt the Prolog convention that all **constant values** in a predicate are either *numeric* or *character strings*; they are represented as identifiers (or names) that start with a *lowercase letter*, whereas **variable names** always start with an *uppercase letter*.

Consider the example shown in Figure 26.11, which is based on the relational database in Figure 3.6, but in a much simplified form. There are three predicate names: *supervise*, *superior*, and *subordinate*. The SUPERVISE predicate is defined via a set of facts, each of which has two arguments: a supervisor name, followed by the name of a *direct* supervisee (subordinate) of that supervisor. These facts correspond to the actual data that is stored in the database, and they can be considered as constituting a set of tuples in a relation SUPERVISE with two attributes whose schema is

SUPERVISE(Supervisor, Supervisee)



**Figure 26.11**  
(a) Prolog notation.  
(b) The supervisory tree.

Thus,  $\text{SUPERVISE}(X, Y)$  states the fact that  $X$  *supervises*  $Y$ . Notice the omission of the attribute names in the Prolog notation. Attribute names are only represented by virtue of the position of each argument in a predicate: the first argument represents the supervisor, and the second argument represents a direct subordinate.

The other two predicate names are defined by rules. The main contributions of deductive databases are the ability to specify recursive rules and to provide a framework for inferring new information based on the specified rules. A rule is of the form **head**  $\text{:-}$  **body**, where  $\text{:-}$  is read as *if and only if*. A rule usually has a single **predicate** to the left of the  $\text{:-}$  symbol—called the **head** or **left-hand side** (LHS) or **conclusion** of the rule—and *one or more* **predicates** to the right of the  $\text{:-}$  symbol—called the **body** or **right-hand side** (RHS) or **premise(s)** of the rule. A predicate with constants as arguments is said to be **ground**; we also refer to it as an **instantiated predicate**. The arguments of the predicates that appear in a rule typically include a number of variable symbols, although predicates can also contain constants as arguments. A rule specifies that, if a particular assignment or **binding** of constant values to the variables in the body (RHS predicates) makes *all* the RHS predicates **true**, it also makes the head (LHS predicate) true by using the same assignment of constant values to variables. Hence, a rule provides us with a way of generating new facts that are instantiations of the head of the rule. These new facts are based on facts that already exist, corresponding to the instantiations (or bindings) of predicates in the body of the rule. Notice that by listing multiple predicates in the body of a rule we implicitly apply the **logical AND** operator to these predicates. Hence, the commas between the RHS predicates may be read as meaning *and*.

Consider the definition of the predicate  $\text{SUPERIOR}$  in Figure 26.11, whose first argument is an employee name and whose second argument is an employee who is either a *direct* or an *indirect* subordinate of the first employee. By *indirect subordinate*, we

mean the subordinate of some subordinate down to any number of levels. Thus  $\text{SUPERIOR}(X, Y)$  stands for the fact that  $X$  is a superior of  $Y$  through direct or indirect supervision. We can write two rules that together specify the meaning of the new predicate. The first rule under Rules in the figure states that for every value of  $X$  and  $Y$ , if  $\text{SUPERVISE}(X, Y)$ —the rule body—is true, then  $\text{SUPERIOR}(X, Y)$ —the rule head—is also true, since  $Y$  would be a direct subordinate of  $X$  (at one level down). This rule can be used to generate all direct superior/subordinate relationships from the facts that define the  $\text{SUPERVISE}$  predicate. The second recursive rule states that if  $\text{SUPERVISE}(X, Z)$  and  $\text{SUPERIOR}(Z, Y)$  are both true, then  $\text{SUPERIOR}(X, Y)$  is also true. This is an example of a **recursive rule**, where one of the rule body predicates in the RHS is the same as the rule head predicate in the LHS. In general, the rule body defines a number of premises such that if they are all true, we can deduce that the conclusion in the rule head is also true. Notice that if we have two (or more) rules with the same head (LHS predicate), it is equivalent to saying that the predicate is true (that is, that it can be instantiated) if *either one* of the bodies is true; hence, it is equivalent to a **logical OR** operation. For example, if we have two rules  $X :- Y$  and  $X :- Z$ , they are equivalent to a rule  $X :- Y \text{ OR } Z$ . The latter form is not used in deductive systems, however, because it is not in the standard form of rule, called a *Horn clause*, as we discuss in Section 26.5.4.

A Prolog system contains a number of **built-in** predicates that the system can interpret directly. These typically include the equality comparison operator  $= (X, Y)$ , which returns true if  $X$  and  $Y$  are identical and can also be written as  $X = Y$  by using the standard infix notation.<sup>31</sup> Other comparison operators for numbers, such as  $<$ ,  $<=$ ,  $>$ , and  $>=$ , can be treated as binary predicates. Arithmetic functions such as  $+$ ,  $-$ ,  $*$ , and  $/$  can be used as arguments in predicates in Prolog. In contrast, Datalog (in its basic form) does *not* allow functions such as arithmetic operations as arguments; indeed, this is one of the main differences between Prolog and Datalog. However, extensions to Datalog have been proposed that do include functions.

A **query** typically involves a predicate symbol with some variable arguments, and its meaning (or *answer*) is to deduce all the different constant combinations that, when **bound** (assigned) to the variables, can make the predicate true. For example, the first query in Figure 26.11 requests the names of all subordinates of *james* at any level. A different type of query, which has only constant symbols as arguments, returns either a true or a false result, depending on whether the arguments provided can be deduced from the facts and rules. For example, the second query in Figure 26.11 returns true, since  $\text{SUPERIOR}(\text{james}, \text{joyce})$  can be deduced.

### 26.5.3 Datalog Notation

In Datalog, as in other logic-based languages, a program is built from basic objects called **atomic formulas**. It is customary to define the syntax of logic-based languages by describing the syntax of atomic formulas and identifying how they can be combined to form a program. In Datalog, atomic formulas are **literals** of the form

<sup>31</sup>A Prolog system typically has a number of different equality predicates that have different interpretations.

$p(a_1, a_2, \dots, a_n)$ , where  $p$  is the predicate name and  $n$  is the number of arguments for predicate  $p$ . Different predicate symbols can have different numbers of arguments, and the number of arguments  $n$  of predicate  $p$  is sometimes called the **arity** or **degree** of  $p$ . The arguments can be either constant values or variable names. As mentioned earlier, we use the convention that constant values either are numeric or start with a *lowercase* character, whereas variable names always start with an *uppercase* character.

A number of **built-in predicates** are included in Datalog and can also be used to construct atomic formulas. The built-in predicates are of two main types: the binary comparison predicates  $<$  (less),  $<=$  (less\_or\_equal),  $>$  (greater), and  $>=$  (greater\_or\_equal) over ordered domains; and the comparison predicates  $=$  (equal) and  $\neq$  (not\_equal) over ordered or unordered domains. These can be used as binary predicates with the same functional syntax as other predicates—for example, by writing `less(X, 3)`—or they can be specified by using the customary infix notation  $X < 3$ . Note that because the domains of these predicates are potentially infinite, they should be used with care in rule definitions. For example, the predicate `greater(X, 3)`, if used alone, generates an infinite set of values for  $X$  that satisfy the predicate (all integer numbers greater than 3).

A **literal** is either an atomic formula as defined earlier—called a **positive literal**—or an atomic formula preceded by **not**. The latter is a negated atomic formula, called a **negative literal**. Datalog programs can be considered to be a *subset* of the predicate calculus formulas, which are somewhat similar to the formulas of the domain relational calculus (see Section 6.7). In Datalog, however, these formulas are first converted into what is known as **clausal form** before they are expressed in Datalog, and only formulas given in a restricted clausal form, called *Horn clauses*,<sup>32</sup> can be used in Datalog.

### 26.5.4 Clausal Form and Horn Clauses

Recall from Section 6.6 that a formula in the relational calculus is a condition that includes predicates called *atoms* (based on relation names). Additionally, a formula can have quantifiers—namely, the *universal quantifier* (for all) and the *existential quantifier* (there exists). In clausal form, a formula must be transformed into another formula with the following characteristics:

- All variables in the formula are universally quantified. Hence, it is not necessary to include the universal quantifiers (for all) explicitly; the quantifiers are removed, and all variables in the formula are *implicitly* quantified by the universal quantifier.
- In clausal form, the formula is made up of a number of clauses, where each **clause** is composed of a number of *literals* connected by OR logical connectives only. Hence, each clause is a *disjunction* of literals.
- The *clauses themselves* are connected by AND logical connectives only, to form a formula. Hence, the **clausal form of a formula** is a *conjunction* of clauses.

<sup>32</sup>Named after the mathematician Alfred Horn.

It can be shown that *any formula can be converted into clausal form*. For our purposes, we are mainly interested in the form of the individual clauses, each of which is a disjunction of literals. Recall that literals can be positive literals or negative literals. Consider a clause of the form:

$$\text{NOT}(P_1) \text{ OR } \text{NOT}(P_2) \text{ OR } \dots \text{ OR } \text{NOT}(P_n) \text{ OR } Q_1 \text{ OR } Q_2 \text{ OR } \dots \text{ OR } Q_m \quad (1)$$

This clause has  $n$  negative literals and  $m$  positive literals. Such a clause can be transformed into the following equivalent logical formula:

$$P_1 \text{ AND } P_2 \text{ AND } \dots \text{ AND } P_n \Rightarrow Q_1 \text{ OR } Q_2 \text{ OR } \dots \text{ OR } Q_m \quad (2)$$

where  $\Rightarrow$  is the **implies** symbol. The formulas (1) and (2) are equivalent, meaning that their truth values are always the same. This is the case because if all the  $P_i$  literals ( $i = 1, 2, \dots, n$ ) are true, the formula (2) is true only if at least one of the  $Q_i$ 's is true, which is the meaning of the  $\Rightarrow$  (implies) symbol. For formula (1), if all the  $P_i$  literals ( $i = 1, 2, \dots, n$ ) are true, their negations are all false; so in this case formula (1) is true only if at least one of the  $Q_i$ 's is true. In Datalog, rules are expressed as a restricted form of clauses called **Horn clauses**, in which a clause can contain *at most one* positive literal. Hence, a Horn clause is either of the form

$$\text{NOT}(P_1) \text{ OR } \text{NOT}(P_2) \text{ OR } \dots \text{ OR } \text{NOT}(P_n) \text{ OR } Q \quad (3)$$

or of the form

$$\text{NOT}(P_1) \text{ OR } \text{NOT}(P_2) \text{ OR } \dots \text{ OR } \text{NOT}(P_n) \quad (4)$$

The Horn clause in (3) can be transformed into the clause

$$P_1 \text{ AND } P_2 \text{ AND } \dots \text{ AND } P_n \Rightarrow Q \quad (5)$$

which is written in Datalog as the following rule:

$$Q :- P_1, P_2, \dots, P_n. \quad (6)$$

The Horn clause in (4) can be transformed into

$$P_1 \text{ AND } P_2 \text{ AND } \dots \text{ AND } P_n \Rightarrow \quad (7)$$

which is written in Datalog as follows:

$$P_1, P_2, \dots, P_n. \quad (8)$$

A **Datalog rule**, as in (6), is hence a Horn clause, and its meaning, based on formula (5), is that if the predicates  $P_1 \text{ AND } P_2 \text{ AND } \dots \text{ AND } P_n$  are all true for a particular binding to their variable arguments, then  $Q$  is also true and can hence be inferred. The Datalog expression (8) can be considered as an integrity constraint, where all the predicates must be true to satisfy the query.

In general, a **query in Datalog** consists of two components:

- A Datalog program, which is a finite set of rules
- A literal  $P(X_1, X_2, \dots, X_n)$ , where each  $X_i$  is a variable or a constant

A Prolog or Datalog system has an internal **inference engine** that can be used to process and compute the results of such queries. Prolog inference engines typically

return one result to the query (that is, one set of values for the variables in the query) at a time and must be prompted to return additional results. On the contrary, Datalog returns results set-at-a-time.

### 26.5.5 Interpretations of Rules

There are two main alternatives for interpreting the theoretical meaning of rules: *proof-theoretic* and *model-theoretic*. In practical systems, the inference mechanism within a system defines the exact interpretation, which may not coincide with either of the two theoretical interpretations. The inference mechanism is a computational procedure and hence provides a computational interpretation of the meaning of rules. In this section, first we discuss the two theoretical interpretations. Then we briefly discuss inference mechanisms as a way of defining the meaning of rules.

In the **proof-theoretic** interpretation of rules, we consider the facts and rules to be true statements, or **axioms**. **Ground axioms** contain no variables. The facts are ground axioms that are given to be true. Rules are called **deductive axioms**, since they can be used to deduce new facts. The deductive axioms can be used to construct proofs that derive new facts from existing facts. For example, Figure 26.12 shows how to prove the fact SUPERIOR(james, ahmad) from the rules and facts given in Figure 26.11. The proof-theoretic interpretation gives us a procedural or computational approach for computing an answer to the Datalog query. The process of proving whether a certain fact (theorem) holds is known as **theorem proving**.

The second type of interpretation is called the **model-theoretic** interpretation. Here, given a finite or an infinite domain of constant values,<sup>33</sup> we assign to a predicate every possible combination of values as arguments. We must then determine whether the predicate is true or false. In general, it is sufficient to specify the combinations of arguments that make the predicate true, and to state that all other combinations make the predicate false. If this is done for every predicate, it is called an **interpretation** of the set of predicates. For example, consider the interpretation shown in Figure 26.13 for the predicates SUPERVISE and SUPERIOR. This interpretation assigns a truth value (true or false) to every possible combination of argument values (from a finite domain) for the two predicates.

An interpretation is called a **model** for a *specific set of rules* if those rules are *always true* under that interpretation; that is, for any values assigned to the variables in the rules, the head of the rules is true when we substitute the truth values assigned to

- 
- |  |                           |
|--|---------------------------|
| 1. SUPERIOR( <i>X</i> , <i>Y</i> ) :- SUPERVISE( <i>X</i> , <i>Y</i> ).                                  | (rule 1)                  |
| 2. SUPERIOR( <i>X</i> , <i>Y</i> ) :- SUPERVISE( <i>X</i> , <i>Z</i> ), SUPERIOR( <i>Z</i> , <i>Y</i> ). | (rule 2)                  |
| 3. SUPERVISE(jennifer, ahmad).   | (ground axiom, given)     |
| 4. SUPERVISE(james, jennifer).   | (ground axiom, given)     |
| 5. SUPERIOR(jennifer, ahmad).  | (apply rule 1 on 3)       |
| 6. SUPERIOR(james, ahmad).   | (apply rule 2 on 4 and 5) |

**Figure 26.12**  
Proving a new fact.

---

<sup>33</sup>The most commonly chosen domain is finite and is called the *Herbrand Universe*.



the predicates in the body of the rule by that interpretation. Hence, whenever a particular substitution (binding) to the variables in the rules is applied, if all the predicates in the body of a rule are true under the interpretation, the predicate in the head of the rule must also be true. The interpretation shown in Figure 26.13 is a model for the two rules shown, since it can never cause the rules to be violated. Notice that a rule is violated if a particular binding of constants to the variables makes all the predicates in the rule body true but makes the predicate in the rule head false. For example, if  $\text{SUPERVISE}(a, b)$  and  $\text{SUPERIOR}(b, c)$  are both true under some interpretation, but  $\text{SUPERIOR}(a, c)$  is not true, the interpretation cannot be a model for the recursive rule:

$$\text{SUPERIOR}(X, Y) :- \text{SUPERVISE}(X, Z), \text{SUPERIOR}(Z, Y)$$

In the model-theoretic approach, the meaning of the rules is established by providing a model for these rules. A model is called a **minimal model** for a set of rules if we cannot change any fact from true to false and still get a model for these rules. For

**Figure 26.13**

An interpretation that is a minimal model.

#### Rules

$\text{SUPERIOR}(X, Y) :- \text{SUPERVISE}(X, Y).$

$\text{SUPERIOR}(X, Y) :- \text{SUPERVISE}(X, Z), \text{SUPERIOR}(Z, Y).$

#### Interpretation

##### *Known Facts:*

$\text{SUPERVISE}(\text{franklin}, \text{john})$  is **true**.

$\text{SUPERVISE}(\text{franklin}, \text{ramesh})$  is **true**.

$\text{SUPERVISE}(\text{franklin}, \text{joyce})$  is **true**.

$\text{SUPERVISE}(\text{jennifer}, \text{alicia})$  is **true**.

$\text{SUPERVISE}(\text{jennifer}, \text{ahmad})$  is **true**.

$\text{SUPERVISE}(\text{james}, \text{franklin})$  is **true**.

$\text{SUPERVISE}(\text{james}, \text{jennifer})$  is **true**.

$\text{SUPERVISE}(X, Y)$  is **false** for all other possible  $(X, Y)$  combinations

##### *Derived Facts:*

$\text{SUPERIOR}(\text{franklin}, \text{john})$  is **true**.

$\text{SUPERIOR}(\text{franklin}, \text{ramesh})$  is **true**.

$\text{SUPERIOR}(\text{franklin}, \text{joyce})$  is **true**.

$\text{SUPERIOR}(\text{jennifer}, \text{alicia})$  is **true**.

$\text{SUPERIOR}(\text{jennifer}, \text{ahmad})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{franklin})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{jennifer})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{john})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{ramesh})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{joyce})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{alicia})$  is **true**.

$\text{SUPERIOR}(\text{james}, \text{ahmad})$  is **true**.

$\text{SUPERIOR}(X, Y)$  is **false** for all other possible  $(X, Y)$  combinations



example, consider the interpretation in Figure 26.13, and assume that the SUPERVISE predicate is defined by a set of known facts, whereas the SUPERIOR predicate is defined as an interpretation (model) for the rules. Suppose that we add the predicate SUPERIOR(james, bob) to the true predicates. This remains a model for the rules shown, but it is not a minimal model, since changing the truth value of SUPERIOR(james, bob) from true to false still provides us with a model for the rules. The model shown in Figure 26.13 is the minimal model for the set of facts that are defined by the SUPERVISE predicate.

In general, the minimal model that corresponds to a given set of facts in the model-theoretic interpretation should be the same as the facts generated by the proof-theoretic interpretation for the same original set of ground and deductive axioms. However, this is generally true only for rules with a simple structure. Once we allow negation in the specification of rules, the correspondence between interpretations *does not* hold. In fact, with negation, numerous minimal models are possible for a given set of facts.

A third approach to interpreting the meaning of rules involves defining an inference mechanism that is used by the system to deduce facts from the rules. This inference mechanism would define a **computational interpretation** to the meaning of the rules. The Prolog logic programming language uses its inference mechanism to define the meaning of the rules and facts in a Prolog program. Not all Prolog programs correspond to the proof-theoretic or model-theoretic interpretations; it depends on the type of rules in the program. However, for many simple Prolog programs, the Prolog inference mechanism infers the facts that correspond either to the proof-theoretic interpretation or to a minimal model under the model-theoretic interpretation.

### 26.5.6 Datalog Programs and Their Safety

There are two main methods of defining the truth values of predicates in actual Datalog programs. **Fact-defined predicates** (or **relations**) are defined by listing all the combinations of values (the tuples) that make the predicate true. These correspond to base relations whose contents are stored in a database system. Figure 26.14 shows the fact-defined predicates EMPLOYEE, MALE, FEMALE, DEPARTMENT, SUPERVISE, PROJECT, and WORKS\_ON, which correspond to part of the relational database shown in Figure 5.6. **Rule-defined predicates** (or **views**) are defined by being the head (LHS) of one or more Datalog rules; they correspond to *virtual relations* whose contents can be inferred by the inference engine. Figure 26.15 shows a number of rule-defined predicates.

A program or a rule is said to be **safe** if it generates a *finite* set of facts. The general theoretical problem of determining whether a set of rules is safe is undecidable. However, one can determine the safety of restricted forms of rules. For example, the rules shown in Figure 26.16 are safe. One situation where we get unsafe rules that can generate an infinite number of facts arises when one of the variables in the rule can range over an infinite domain of values, and that variable is not limited to ranging over a finite relation. For example, consider the following rule:

```
BIG_SALARY(Y) :- Y > 60000
```

**Figure 26.14**

Fact-defined  
predicates for part  
of the database from  
Figure 5.6.

EMPLOYEE(john).	MALE(john).
EMPLOYEE(franklin).	MALE(franklin).
EMPLOYEE(alicia).	MALE(ramesh).
EMPLOYEE(jennifer).	MALE(ahmad).
EMPLOYEE(ramesh).	MALE(james).
EMPLOYEE(joyce).	
EMPLOYEE(ahmad).	FEMALE(alicia).
EMPLOYEE(james).	FEMALE(jennifer).
	FEMALE(joyce).
SALARY(john, 30000).	
SALARY(franklin, 40000).	PROJECT(productx).
SALARY(alicia, 25000).	PROJECT(producty).
SALARY(jennifer, 43000).	PROJECT(productz).
SALARY(ramesh, 38000).	PROJECT(computerization).
SALARY(joyce, 25000).	PROJECT(reorganization).
SALARY(ahmad, 25000).	PROJECT(newbenefits).
SALARY(james, 55000).	
DEPARTMENT(john, research).	WORKS_ON(john, productx, 32).
DEPARTMENT(franklin, research).	WORKS_ON(john, producty, 8).
DEPARTMENT(alicia, administration).	WORKS_ON(ramesh, productz, 40).
DEPARTMENT(jennifer, administration).	WORKS_ON(joyce, productx, 20).
DEPARTMENT(ramesh, research).	WORKS_ON(joyce, producty, 20).
DEPARTMENT(joyce, research).	WORKS_ON(franklin, producty, 10).
DEPARTMENT(ahmad, administration).	WORKS_ON(franklin, productz, 10).
DEPARTMENT(james, headquarters).	WORKS_ON(franklin, computerization, 10).
	WORKS_ON(franklin, reorganization, 10).
SUPERVISE(franklin, john).	WORKS_ON(alicia, newbenefits, 30).
SUPERVISE(franklin, ramesh).	WORKS_ON(alicia, computerization, 10).
SUPERVISE(franklin, joyce).	WORKS_ON(ahmad, computerization, 35).
SUPERVISE(jennifer, alicia).	WORKS_ON(ahmad, newbenefits, 5).
SUPERVISE(jennifer, ahmad).	WORKS_ON(jennifer, newbenefits, 20).
SUPERVISE(james, franklin).	WORKS_ON(jennifer, reorganization, 15).
SUPERVISE(james, jennifer).	WORKS_ON(james, reorganization, 10).

**Figure 26.15**

Rule-defined  
predicates.

SUPERIOR( $X, Y$ ) :- SUPERVISE( $X, Y$ ).  
 SUPERIOR( $X, Y$ ) :- SUPERVISE( $X, Z$ ), SUPERIOR( $Z, Y$ ).  
 SUBORDINATE( $X, Y$ ) :- SUPERIOR( $Y, X$ ).  
 SUPERVISOR( $X$ ) :- EMPLOYEE( $X$ ), SUPERVISE( $X, Y$ ).  
 OVER\_40K\_EMP( $X$ ) :- EMPLOYEE( $X$ ), SALARY( $X, Y$ ),  $Y \geq 40000$ .  
 UNDER\_40K\_SUPERVISOR( $X$ ) :- SUPERVISOR( $X$ ), NOT(OVER\_40K\_EMP( $X$ )).  
 MAIN\_PRODUCTX\_EMP( $X$ ) :- EMPLOYEE( $X$ ), WORKS\_ON( $X$ , productx,  $Y$ ),  $Y \geq 20$ .  
 PRESIDENT( $X$ ) :- EMPLOYEE( $X$ ), NOT(SUPERVISE( $Y, X$ )).

```

REL_ONE(A, B, C).
REL_TWO(D, E, F).
REL_THREE(G, H, I, J).

SELECT_ONE_A_EQ_C(X, Y, Z) :- REL_ONE(C, Y, Z).
SELECT_ONE_B_LESS_5(X, Y, Z) :- REL_ONE(X, Y, Z), Y < 5.
SELECT_ONE_A_EQ_C_AND_B_LESS_5(X, Y, Z) :- REL_ONE(C, Y, Z), Y < 5.

SELECT_ONE_A_EQ_C_OR_B_LESS_5(X, Y, Z) :- REL_ONE(C, Y, Z).
SELECT_ONE_A_EQ_C_OR_B_LESS_5(X, Y, Z) :- REL_ONE(X, Y, Z), Y < 5.

PROJECT_THREE_ON_G_H(W, X) :- REL_THREE(W, X, Y, Z).

UNION_ONE_TWO(X, Y, Z) :- REL_ONE(X, Y, Z).
UNION_ONE_TWO(X, Y, Z) :- REL_TWO(X, Y, Z).

INTERSECT_ONE_TWO(X, Y, Z) :- REL_ONE(X, Y, Z), REL_TWO(X, Y, Z).

DIFFERENCE_TWO_ONE(X, Y, Z) :- _TWO(X, Y, Z) NOT(REL_ONE(X, Y, Z)).

CART_PROD_ONE_THREE(T, U, V, W, X, Y, Z) :-
    REL_ONE(T, U, V), REL_THREE(W, X, Y, Z).

NATURAL_JOIN_ONE_THREE_C_EQ_G(U, V, W, X, Y, Z) :-
    REL_ONE(U, V, W), REL_THREE(W, X, Y, Z).

```

**Figure 26.16**

Predicates for illustrating relational operations.

Here, we can get an infinite result if  $Y$  ranges over all possible integers. But suppose that we change the rule as follows:

```
BIG_SALARY(Y) :- EMPLOYEE(X), Salary(X, Y), Y > 60000
```

In the second rule, the result is not infinite, since the values that  $Y$  can be bound to are now restricted to values that are the salary of some employee in the database—presumably, a finite set of values. We can also rewrite the rule as follows:

```
BIG_SALARY(Y) :- Y > 60000, EMPLOYEE(X), Salary(X, Y)
```

In this case, the rule is still theoretically safe. However, in Prolog or any other system that uses a top-down, depth-first inference mechanism, the rule creates an infinite loop, since we first search for a value for  $Y$  and then check whether it is a salary of an employee. The result is generation of an infinite number of  $Y$  values, even though these, after a certain point, cannot lead to a set of true RHS predicates. One definition of Datalog considers both rules to be safe, since it does not depend on a particular inference mechanism. Nonetheless, it is generally advisable to write such a rule in the safest form, with the predicates that restrict possible bindings of variables placed first. As another example of an unsafe rule, consider the following rule:

```
HAS_SOMETHING(X, Y) :- EMPLOYEE(X)
```

Here, an infinite number of  $Y$  values can again be generated, since the variable  $Y$  appears only in the head of the rule and hence is not limited to a finite set of values. To define safe rules more formally, we use the concept of a limited variable. A variable  $X$  is **limited** in a rule if (1) it appears in a regular (not built-in) predicate in the body of the rule; (2) it appears in a predicate of the form  $X = c$  or  $c = X$  or  $(c_1 \leq X$  and  $X \leq c_2)$  in the rule body, where  $c$ ,  $c_1$ , and  $c_2$  are constant values; or (3) it appears in a predicate of the form  $X = Y$  or  $Y = X$  in the rule body, where  $Y$  is a limited variable. A rule is said to be **safe** if all its variables are limited.

### 26.5.7 Use of Relational Operations

It is straightforward to specify many operations of the relational algebra in the form of Datalog rules that define the result of applying these operations on the database relations (fact predicates). This means that relational queries and views can easily be specified in Datalog. The additional power that Datalog provides is in the specification of recursive queries, and views based on recursive queries. In this section, we show how some of the standard relational operations can be specified as Datalog rules. Our examples will use the base relations (fact-defined predicates) `REL_ONE`, `REL_TWO`, and `REL_THREE`, whose schemas are shown in Figure 26.16. In Datalog, we do not need to specify the attribute names as in Figure 26.16; rather, the arity (degree) of each predicate is the important aspect. In a practical system, the domain (data type) of each attribute is also important for operations such as `UNION`, `INTERSECTION`, and `JOIN`, and we assume that the attribute types are compatible for the various operations, as discussed in Chapter 3.

Figure 26.16 illustrates a number of basic relational operations. Notice that if the Datalog model is based on the relational model and hence assumes that predicates (fact relations and query results) specify sets of tuples, duplicate tuples in the same predicate are automatically eliminated. This may or may not be true, depending on the Datalog inference engine. However, it is definitely *not* the case in Prolog, so any of the rules in Figure 26.16 that involve duplicate elimination are not correct for Prolog. For example, if we want to specify Prolog rules for the `UNION` operation with duplicate elimination, we must rewrite them as follows:

```
UNION_ONE_TWO(X, Y, Z) :- REL_ONE(X, Y, Z).
UNION_ONE_TWO(X, Y, Z) :- REL_TWO(X, Y, Z), NOT(REL_ONE(X, Y, Z)).
```

However, the rules shown in Figure 26.16 should work for Datalog, if duplicates are automatically eliminated. Similarly, the rules for the `PROJECT` operation shown in Figure 26.16 should work for Datalog in this case, but they are not correct for Prolog, since duplicates would appear in the latter case.

### 26.5.8 Evaluation of Nonrecursive Datalog Queries

In order to use Datalog as a deductive database system, it is appropriate to define an inference mechanism based on relational database query processing concepts. The inherent strategy involves a bottom-up evaluation, starting with base relations; the order of operations is kept flexible and subject to query optimization. In this section we discuss an **inference mechanism** based on relational operations that can be

applied to **nonrecursive** Datalog queries. We use the fact and rule base shown in Figures 26.14 and 26.15 to illustrate our discussion.

If a query involves only fact-defined predicates, the inference becomes one of searching among the facts for the query result. For example, a query such as

DEPARTMENT( $X$ , Research)?

is a selection of all employee names  $X$  who work for the Research department. In relational algebra, it is the query:

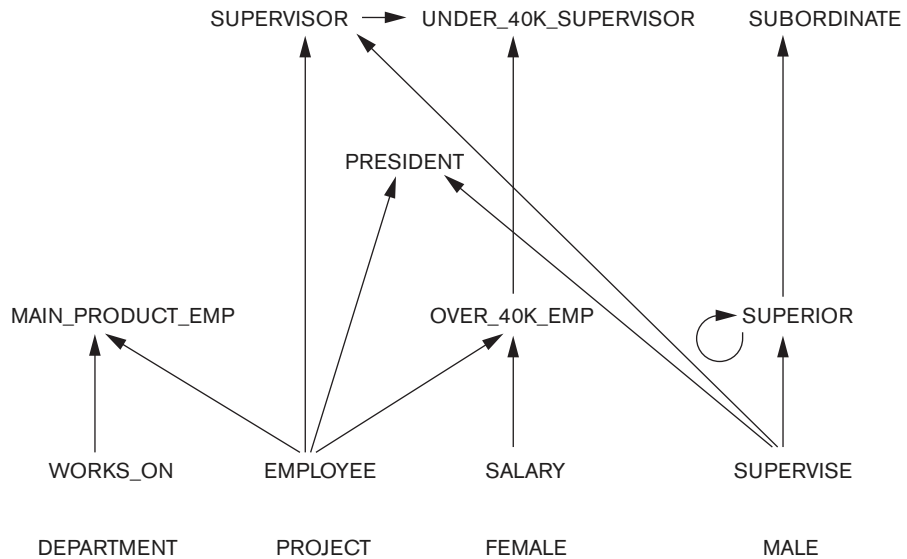
$\pi_{\$1} (\sigma_{\$2 = \text{"Research"}} (\text{DEPARTMENT}))$

which can be answered by searching through the fact-defined predicate `department( $X$ ,  $Y$ )`. The query involves relational SELECT and PROJECT operations on a base relation, and it can be handled by the database query processing and optimization techniques discussed in Chapter 19.

When a query involves rule-defined predicates, the inference mechanism must compute the result based on the rule definitions. If a query is nonrecursive and involves a predicate  $p$  that appears as the head of a rule  $p :- p_1, p_2, \dots, p_n$ , the strategy is first to compute the relations corresponding to  $p_1, p_2, \dots, p_n$  and then to compute the relation corresponding to  $p$ . It is useful to keep track of the dependency among the predicates of a deductive database in a **predicate dependency graph**. Figure 26.17 shows the graph for the fact and rule predicates shown in Figures 26.14 and 26.15. The dependency graph contains a **node** for each predicate. Whenever a predicate  $A$  is specified in the body (RHS) of a rule, and the head (LHS) of that rule is the predicate  $B$ , we say that  $B$  **depends on**  $A$ , and we draw a directed edge from  $A$  to  $B$ . This indicates that in order to compute the facts for the predicate  $B$  (the rule head), we must first compute the facts for all the predicates  $A$  in the rule body. If the dependency graph has no cycles, we call the rule set **nonrecursive**. If there is at least one cycle, we call the rule set **recursive**. In Figure 26.17, there is one recursively defined predicate—namely, `SUPERIOR`—which has a recursive edge pointing back to itself. Additionally, because the predicate `subordinate` depends on `SUPERIOR`, it also requires recursion in computing its result.

A query that includes only nonrecursive predicates is called a **nonrecursive query**. In this section we discuss only inference mechanisms for nonrecursive queries. In Figure 26.17, any query that does not involve the predicates `SUBORDINATE` or `SUPERIOR` is nonrecursive. In the predicate dependency graph, the nodes corresponding to fact-defined predicates do not have any incoming edges, since all fact-defined predicates have their facts stored in a database relation. The contents of a fact-defined predicate can be computed by directly retrieving the tuples in the corresponding database relation.

The main function of an inference mechanism is to compute the facts that correspond to query predicates. This can be accomplished by generating a **relational expression** involving relational operators as SELECT, PROJECT, JOIN, UNION, and SET DIFFERENCE (with appropriate provision for dealing with safety issues) that, when executed, provides the query result. The query can then be executed by utilizing the internal query processing and optimization operations of a relational database



**Figure 26.17**  
 Predicate dependency  
 graph for Figures 26.15  
 and 26.16.

management system. Whenever the inference mechanism needs to compute the fact set corresponding to a nonrecursive rule-defined predicate  $p$ , it first locates all the rules that have  $p$  as their head. The idea is to compute the fact set for each such rule and then to apply the UNION operation to the results, since UNION corresponds to a logical OR operation. The dependency graph indicates all predicates  $q$  on which each  $p$  depends, and since we assume that the predicate is nonrecursive, we can always determine a partial order among such predicates  $q$ . Before computing the fact set for  $p$ , first we compute the fact sets for all predicates  $q$  on which  $p$  depends, based on their partial order. For example, if a query involves the predicate UNDER\_40K\_SUPERVISOR, we must first compute both SUPERVISOR and OVER\_40K\_EMP. Since the latter two depend only on the fact-defined predicates EMPLOYEE, SALARY, and SUPERVISE, they can be computed directly from the stored database relations.

This concludes our introduction to deductive databases. Additional material may be found at the book's Web site, where the complete Chapter 25 from the third edition is available. Information on the Web site includes a discussion on algorithms for recursive query processing. We have included an extensive bibliography of work in deductive databases, recursive query processing, magic sets, combination of relational databases with deductive rules, and GLUE-NAIL! System, at the end of this chapter.

## 26.6 Summary

In this chapter, we introduced database concepts for some of the common features that are needed by advanced applications: active databases, temporal databases, spatial databases, multimedia databases, and deductive databases. It is important to note that each of these is a broad topic and warrants a complete textbook.

First in Section 26.1 we introduced the topic of active databases, which provide additional functionality for specifying active rules. We introduced the event-condition-action (ECA) model for active databases. The rules can be automatically triggered by events that occur—such as a database update—and they can initiate certain actions that have been specified in the rule declaration if certain conditions are true. Many commercial packages have some of the functionality provided by active databases in the form of triggers. We gave examples of row-level triggers in the Oracle commercial system in Section 26.1.1. We discussed the different options for specifying triggers in Section 26.1.2, such as row-level versus statement-level, before versus after, and immediate versus deferred. Then in Section 26.1.3 we gave examples of statement-level rules in the STARBURST experimental system. We briefly discussed some design issues and some possible applications for active databases in Section 26.1.4. The syntax for triggers in the SQL-99 standard was also discussed in Section 26.1.5.

Next in Section 26.2 we introduced some of the concepts of temporal databases, which permit the database system to store a history of changes and allow users to query both current and past states of the database. In Section 26.2.1, we discussed how time is represented and distinguished between the valid time and transaction time dimensions. In Section 26.2.2 we discussed how valid time, transaction time, and bitemporal relations can be implemented using tuple versioning in the relational model, and we provided examples to illustrate how updates, inserts, and deletes are implemented. We also showed how complex objects can be used to implement temporal databases using attribute versioning in Section 26.2.3. We looked at some of the querying operations for temporal relational databases and gave a brief introduction to the TSQL2 language in Section 26.2.4.

Then we turned to spatial databases in Section 26.3. Spatial databases provide concepts for databases that keep track of objects that have spatial characteristics. We gave an introduction to spatial databases in Section 26.3.1. We discussed the types of spatial data and spatial data models in Section 26.3.2, then the types of operators for processing spatial data and the types of spatial queries in Section 26.3.3. In Section 26.3.4, we gave an overview of spatial indexing techniques, including the popular *R*-trees. Then we introduced some spatial data mining techniques in Section 26.3.5, and discussed some applications that require spatial databases in Section 26.3.6.

In Section 26.4 we discussed some basic types of multimedia databases and their important characteristics. Multimedia databases provide features that allow users to store and query different types of multimedia information, which includes images (such as pictures and drawings), video clips (such as movies, newsreels, and home videos), audio clips (such as songs, phone messages, and speeches), and documents (such as books and articles). We provided a brief overview of the various types of media sources and how multimedia sources may be indexed. Images are an extremely common type of data among databases today and are likely to occupy a large proportion of stored data in databases. We therefore provided a more detailed treatment of images: their automatic analysis (Section 26.4.1), recognition of objects within images (Section 26.4.2), and their semantic tagging (Section 26.1.3)—all of which contribute to developing better systems to retrieve images by content, which



still remains a challenging problem. We also commented on the analysis of audio data sources in Section 26.4.4.

We concluded the chapter with an introduction to deductive databases in Section 26.5. We introduced deductive databases in Section 26.5.1, and gave an overview of Prolog and Datalog notation in Sections 26.5.2 and 26.5.3. We discussed the clausal form of formulas in Section 26.5.4. Datalog rules are restricted to Horn clauses, which contain at most one positive literal. We discussed the proof-theoretic and model-theoretic interpretation of rules in Section 26.5.5. We briefly discussed the safety of Datalog rules in Section 26.5.6 and the ways of expressing relational operators using Datalog rules in Section 26.5.7. Finally, we discussed an inference mechanism based on relational operations that can be used to evaluate nonrecursive Datalog queries using relational query optimization techniques in Section 26.5.8. Although Datalog has been a popular language with some applications, implementations of deductive database systems such as LDL or VALIDITY have not become widely commercially available.

## Review Questions

- 26.1.** What are the differences between row-level and statement-level active rules?
- 26.2.** What are the differences among immediate, deferred, and detached *consideration* of active rule conditions?
- 26.3.** What are the differences among immediate, deferred, and detached *execution* of active rule actions?
- 26.4.** Briefly discuss the consistency and termination problems when designing a set of active rules.
- 26.5.** Discuss some applications of active databases.
- 26.6.** Discuss how time is represented in temporal databases and compare the different time dimensions.
- 26.7.** What are the differences among valid time, transaction time, and bitemporal relations?
- 26.8.** Describe how the insert, delete, and update commands should be implemented on a valid time relation.
- 26.9.** Describe how the insert, delete, and update commands should be implemented on a bitemporal relation.
- 26.10.** Describe how the insert, delete, and update commands should be implemented on a transaction time relation.
- 26.11.** What are the main differences between tuple versioning and attribute versioning?
- 26.12.** How do spatial databases differ from regular databases?
- 26.13.** What are the different types of spatial data?



- 26.14. Name the main types of spatial operators and different classes of spatial queries.
- 26.15. What are the properties of *R*-trees that act as an index for spatial data?
- 26.16. Describe how a spatial join index between spatial objects can be constructed.
- 26.17. What are the different types of spatial data mining?
- 26.18. State the general form of a spatial association rule. Give an example of a spatial association rule.
- 26.19. What are the different types of multimedia sources?
- 26.20. How are multimedia sources indexed for content-based retrieval?
- 26.21. What important features of images are used to compare them?
- 26.22. What are the different approaches to recognizing objects in images?
- 26.23. How is semantic tagging of images used?
- 26.24. What are the difficulties in analyzing audio sources?
- 26.25. What are deductive databases?
- 26.26. Write sample rules in Prolog to define that courses with course number above CS5000 are graduate courses and that DBgrads are those graduate students who enroll in CS6400 and CS8803.
- 26.27. Define the clausal form of formulas and Horn clauses.
- 26.28. What is theorem proving, and what is proof-theoretic interpretation of rules?
- 26.29. What is model-theoretic interpretation and how does it differ from proof-theoretic interpretation?
- 26.30. What are fact-defined predicates and rule-defined predicates?
- 26.31. What is a safe rule?
- 26.32. Give examples of rules that can define relational operations SELECT, PROJECT, JOIN, and SET operations.
- 26.33. Discuss the inference mechanism based on relational operations that can be applied to evaluate nonrecursive Datalog queries.

## Exercises

- 26.34. Consider the COMPANY database described in Figure 5.6. Using the syntax of Oracle triggers, write active rules to do the following:
  - a. Whenever an employee's project assignments are changed, check if the total hours per week spent on the employee's projects are less than 30 or greater than 40; if so, notify the employee's direct supervisor.

**SALES**

<u>S_id</u>	<u>V_id</u>	Commission
-------------	-------------	------------

**Figure 26.18**

Database schema for sales and salesperson commissions in Exercise 26.36.

**SALES\_PERSON**

<u>Salesperson_id</u>	Name	Title	Phone	Sum_commissions
-----------------------	------	-------	-------	-----------------

- b. Whenever an employee is deleted, delete the PROJECT tuples and DEPENDENT tuples related to that employee, and if the employee manages a department or supervises employees, set the Mgr\_ssn for that department to NULL and set the Super\_ssn for those employees to NULL.
- 26.35.** Repeat Exercise 26.34 but use the syntax of STARBURST active rules.
- 26.36.** Consider the relational schema shown in Figure 26.18. Write active rules for keeping the Sum\_commissions attribute of SALES\_PERSON equal to the sum of the Commission attribute in SALES for each sales person. Your rules should also check if the Sum\_commissions exceeds 100000; if it does, call a procedure Notify\_manager(S\_id). Write both statement-level rules in STARBURST notation and row-level rules in Oracle.
- 26.37.** Consider the UNIVERSITY EER schema in Figure 4.10. Write some rules (in English) that could be implemented via active rules to enforce some common integrity constraints that you think are relevant to this application.
- 26.38.** Discuss which of the updates that created each of the tuples shown in Figure 26.9 were applied retroactively and which were applied proactively.
- 26.39.** Show how the following updates, if applied in sequence, would change the contents of the bitemporal EMP\_BT relation in Figure 26.9. For each update, state whether it is a retroactive or proactive update.
- a. On 2004-03-10,17:30:00, the salary of Narayan is updated to 40000, effective on 2004-03-01.
  - b. On 2003-07-30,08:31:00, the salary of Smith was corrected to show that it should have been entered as 31000 (instead of 30000 as shown), effective on 2003-06-01.
  - c. On 2004-03-18,08:31:00, the database was changed to indicate that Narayan was leaving the company (that is, logically deleted) effective on 2004-03-31.
  - d. On 2004-04-20,14:07:33, the database was changed to indicate the hiring of a new employee called Johnson, with the tuple <'Johnson', '334455667', 1, NULL > effective on 2004-04-20.
  - e. On 2004-04-28,12:54:02, the database was changed to indicate that Wong was leaving the company (that is, logically deleted) effective on 2004-06-01.
  - f. On 2004-05-05,13:07:33, the database was changed to indicate the rehiring of Brown, with the same department and supervisor but with salary 35000 effective on 2004-05-01.

**26.40.** Show how the updates given in Exercise 26.39, if applied in sequence, would change the contents of the valid time EMP\_VT relation in Figure 26.8.

**26.41.** Add the following facts to the sample database in Figure 26.11:

SUPERVISE(ahmad, bob), SUPERVISE(franklin, gwen)

First modify the supervisory tree in Figure 26.11(b) to reflect this change. Then construct a diagram showing the top-down evaluation of the query SUPERIOR(james, Y) using rules 1 and 2 from Figure 26.12.

**26.42.** Consider the following set of facts for the relation PARENT(X, Y), where Y is the parent of X:

PARENT(a, aa), PARENT(a, ab), PARENT(aa, aaa), PARENT(aa, aab),  
PARENT(aaa, aaaa), PARENT(aaa, aaab)

Consider the rules

$r_1$ : ANCESTOR(X, Y) :- PARENT(X, Y)

$r_2$ : ANCESTOR(X, Y) :- PARENT(X, Z), ANCESTOR(Z, Y)

which define ancestor Y of X as above.

a. Show how to solve the Datalog query

ANCESTOR(aa, X)?

and show your work at each step.

b. Show the same query by computing only the changes in the ancestor relation and using that in rule 2 each time.

[This question is derived from Bancilhon and Ramakrishnan (1986).]

**26.43.** Consider a deductive database with the following rules:

ANCESTOR(X, Y) :- FATHER(X, Y)

ANCESTOR(X, Y) :- FATHER(X, Z), ANCESTOR(Z, Y)

Notice that FATHER(X, Y) means that Y is the father of X; ANCESTOR(X, Y) means that Y is the ancestor of X.

Consider the following fact base:

FATHER(Harry, Issac), FATHER(Issac, John), FATHER(John, Kurt)

a. Construct a model-theoretic interpretation of the above rules using the given facts.

b. Consider that a database contains the above relations FATHER(X, Y), another relation BROTHER(X, Y), and a third relation BIRTH(X, B), where B is the birth date of person X. State a rule that computes the first cousins of the following variety: their fathers must be brothers.

c. Show a complete Datalog program with fact-based and rule-based literals that computes the following relation: list of pairs of cousins, where the first person is born after 1960 and the second after 1970. You may use *greater than* as a built-in predicate. (Note: Sample facts for brother, birth, and person must also be shown.)

**26.44.** Consider the following rules:

$\text{REACHABLE}(X, Y) :- \text{FLIGHT}(X, Y)$

$\text{REACHABLE}(X, Y) :- \text{FLIGHT}(X, Z), \text{REACHABLE}(Z, Y)$

where  $\text{REACHABLE}(X, Y)$  means that city  $Y$  can be reached from city  $X$ , and  $\text{FLIGHT}(X, Y)$  means that there is a flight to city  $Y$  from city  $X$ .

a. Construct fact predicates that describe the following:

Los Angeles, New York, Chicago, Atlanta, Frankfurt, Paris, Singapore, Sydney are cities.

The following flights exist: LA to NY, NY to Atlanta, Atlanta to Frankfurt, Frankfurt to Atlanta, Frankfurt to Singapore, and Singapore to Sydney. (*Note:* No flight in reverse direction can be automatically assumed.)

b. Is the given data cyclic? If so, in what sense?

c. Construct a model-theoretic interpretation (that is, an interpretation similar to the one shown in Figure 26.13) of the above facts and rules.

d. Consider the query

$\text{REACHABLE}(\text{Atlanta}, \text{Sydney})?$

How will this query be executed? List the series of steps it will go through.

e. Consider the following rule-defined predicates:

$\text{ROUND-TRIP-REACHABLE}(X, Y) :-$

$\text{REACHABLE}(X, Y), \text{REACHABLE}(Y, X)$

$\text{DURATION}(X, Y, Z)$

Draw a predicate dependency graph for the above predicates. (*Note:*  $\text{DURATION}(X, Y, Z)$  means that you can take a flight from  $X$  to  $Y$  in  $Z$  hours.)

f. Consider the following query: What cities are reachable in 12 hours from Atlanta? Show how to express it in Datalog. Assume built-in predicates like  $\text{greater-than}(X, Y)$ . Can this be converted into a relational algebra statement in a straightforward way? Why or why not?

g. Consider the predicate  $\text{population}(X, Y)$ , where  $Y$  is the population of city  $X$ . Consider the following query: List all possible bindings of the predicate pair  $(X, Y)$ , where  $Y$  is a city that can be reached in two flights from city  $X$ , which has over 1 million people. Show this query in Datalog. Draw a corresponding query tree in relational algebraic terms.

## Selected Bibliography

The book by Zaniolo et al. (1997) consists of several parts, each describing an advanced database concept such as active, temporal, and spatial/text/multimedia databases. Widom and Ceri (1996) and Ceri and Fraternali (1997) focus on active database concepts and systems. Snodgrass (1995) describes the TSQL2 language and data model. Khoshafian and Baker (1996), Faloutsos (1996), and Subrahmanian (1998) describe multimedia database concepts. Tansel et al. (1993) is a collection of chapters on temporal databases. The temporal extensions to SQL:2011 are discussed in Kulkarni and Michels (2012).

STARBURST rules are described in Widom and Finkelstein (1990). Early work on active databases includes the HiPAC project, discussed in Chakravarthy et al. (1989) and Chakravarthy (1990). A glossary for temporal databases is given in Jensen et al. (1994). Snodgrass (1987) focuses on TQuel, an early temporal query language.

Temporal normalization is defined in Navathe and Ahmed (1989). Paton (1999) and Paton and Diaz (1999) survey active databases. Chakravarthy et al. (1994) describe SENTINEL and object-based active systems. Lee et al. (1998) discuss time series management.

The book by Shekhar and Chawla (2003) consists of all aspects of spatial databases including spatial data models, spatial storage and indexing, and spatial data mining. Scholl et al. (2001) is another textbook on spatial data management. Albrecht (1996) describes in detail the various GIS analysis operations. Clementini and Di Felice (1993) give a detailed description of the spatial operators. Güting (1994) describes the spatial data structures and querying languages for spatial database systems. Guttman (1984) proposed R-trees for spatial data indexing. Manolopoulos et al. (2005) is a book on the theory and applications of R-trees. Papadias et al. (2003) discuss query processing using R-trees for spatial networks. Ester et al. (2001) provide a comprehensive discussion on the algorithms and applications of spatial data mining. Koperski and Han (1995) discuss association rule discovery from geographic databases. Brinkhoff et al. (1993) provide a comprehensive overview of the usage of R-trees for efficient processing of spatial joins. Rotem (1991) describes spatial join indexes comprehensively. Shekhar and Xiong (2008) is a compilation of various sources that discuss different aspects of spatial database management systems and GIS. The density-based clustering algorithms DBSCAN and DENCLUE are proposed by Ester et al. (1996) and Hinnenberg and Gabriel (2007), respectively.

Multimedia database modeling has a vast amount of literature—it is difficult to point to all important references here. IBM's QBIC (Query By Image Content) system described in Niblack et al. (1998) was one of the first comprehensive approaches for querying images based on content. It is now available as a part of IBM's DB2 database image extender. Zhao and Grosky (2002) discuss content-based image retrieval. Carneiro and Vasconcelos (2005) present a database-centric view of semantic image annotation and retrieval. Content-based retrieval of subimages is discussed by Luo and Nascimento (2004). Tuceryan and Jain (1998) discuss various aspects of texture analysis. Object recognition using SIFT is discussed in Lowe (2004). Lazebnik et al. (2004) describe the use of local affine regions to model 3D objects (RIFT). Among other object recognition approaches, G-RIF is described in Kim et al. (2006). Bay et al. (2006) discuss SURF, Ke and Sukthankar (2004) present PCA-SIFT, and Mikolajczyk and Schmid (2005) describe GLOH. Fan et al. (2004) present a technique for automatic image annotation by using concept-sensitive objects. Fotouhi et al. (2007) was the first international workshop on many faces of multimedia semantics, which is continuing annually. Thuraisingham (2001) classifies audio data into different categories and, by treating each of these categories differently, elaborates on the use of metadata for audio. Prabhakaran (1996) has also discussed how speech processing techniques can add valuable metadata information to the audio piece.

The early developments of the logic and database approach are surveyed by Gallaire et al. (1984). Reiter (1984) provides a reconstruction of relational database theory,

whereas Levesque (1984) provides a discussion of incomplete knowledge in light of logic. Gallaire and Minker (1978) provide an early book on this topic. A detailed treatment of logic and databases appears in Ullman (1989, Volume 2), and there is a related chapter in Volume 1 (1988). Ceri, Gottlob, and Tanca (1990) present a comprehensive yet concise treatment of logic and databases. Das (1992) is a comprehensive book on deductive databases and logic programming. The early history of Datalog is covered in Maier and Warren (1988). Clocksin and Mellish (2003) is an excellent reference on Prolog language.

Aho and Ullman (1979) provide an early algorithm for dealing with recursive queries, using the least fixed-point operator. Bancilhon and Ramakrishnan (1986) give an excellent and detailed description of the approaches to recursive query processing, with detailed examples of the naive and seminaive approaches. Excellent survey articles on deductive databases and recursive query processing include Warren (1992) and Ramakrishnan and Ullman (1995). A complete description of the seminaive approach based on relational algebra is given in Bancilhon (1985). Other approaches to recursive query processing include the recursive query/subquery strategy of Vieille (1986), which is a top-down interpreted strategy, and the Henschen-Naqvi (1984) top-down compiled iterative strategy. Balbin and Ramamohanrao (1987) discuss an extension of the seminaive differential approach for multiple predicates.

The original paper on magic sets is by Bancilhon et al. (1986). Beeri and Ramakrishnan (1987) extend it. Mumick et al. (1990a) show the applicability of magic sets to nonrecursive nested SQL queries. Other approaches to optimizing rules without rewriting them appear in Vieille (1986, 1987). Kifer and Lozinskii (1986) propose a different technique. Bry (1990) discusses how the top-down and bottom-up approaches can be reconciled. Whang and Navathe (1992) describe an extended disjunctive normal form technique to deal with recursion in relational algebra expressions for providing an expert system interface over a relational DBMS.

Chang (1981) describes an early system for combining deductive rules with relational databases. The LDL system prototype is described in Chimenti et al. (1990). Krishnamurthy and Naqvi (1989) introduce the *choice* notion in LDL. Zaniolo (1988) discusses the language issues for the LDL system. A language overview of CORAL is provided in Ramakrishnan et al. (1992), and the implementation is described in Ramakrishnan et al. (1993). An extension to support object-oriented features, called CORAL++, is described in Srivastava et al. (1993). Ullman (1985) provides the basis for the NAIL! system, which is described in Morris et al. (1987). Phipps et al. (1991) describe the GLUE-NAIL! deductive database system.

Zaniolo (1990) reviews the theoretical background and the practical importance of deductive databases. Nicolas (1997) gives an excellent history of the developments leading up to deductive object-oriented database (DOOD) systems. Falcone et al. (1997) survey the DOOD landscape. References on the VALIDITY system include Friesen et al. (1995), Vieille (1998), and Dietrich et al. (1999).

## Introduction to Information Retrieval and Web Search

In most of the chapters in this book so far, we have discussed techniques for modeling, designing, querying, transaction processing of, and managing *structured data*. In Section 13.1, we discussed the differences among structured, semistructured, and unstructured data. Information retrieval deals mainly with *unstructured data*, and the techniques for indexing, searching, and retrieving information from large collections of unstructured documents. In Chapter 24, on NOSQL technologies, we considered systems, like MongoDB, that are suited to handling data in the form of documents. In this chapter,<sup>1</sup> we will provide an introduction to information retrieval. This is a very broad topic, so we will focus on the similarities and differences between information retrieval and database technologies, and on the indexing techniques that form the basis of many information retrieval systems.

This chapter is organized as follows. In Section 27.1, we introduce information retrieval (IR) concepts and discuss how IR differs from traditional databases. Section 27.2 is devoted to a discussion of retrieval models, which form the basis for IR search. Section 27.3 covers different types of queries in IR systems. Section 27.4 discusses text preprocessing, and Section 27.5 provides an overview of IR indexing, which is at the heart of any IR system. In Section 27.6, we describe the various evaluation metrics for IR systems performance. Section 27.7 details Web analysis and its relationship to information retrieval, and Section 27.8 briefly introduces the current trends in IR. Section 27.9 summarizes the chapter. For a limited overview of IR, we suggest that students read Sections 27.1 through 27.6.

---

<sup>1</sup>This chapter is coauthored with Saurav Sahay, Intel Labs.



## 27.1 Information Retrieval (IR) Concepts

**Information retrieval** is the process of retrieving documents from a collection in response to a query (or a search request) by a user. This section provides an overview of IR concepts. In Section 27.1.1, we introduce information retrieval in general and then discuss the different kinds and levels of search that IR encompasses. In Section 27.1.2, we compare IR and database technologies. Section 27.1.3 gives a brief history of IR. We then present the different modes of user interaction with IR systems in Section 27.1.4. In Section 27.1.5, we describe the typical IR process with a detailed set of tasks and then with a simplified process flow, and we end with a brief discussion of digital libraries and the Web.

### 27.1.1 Introduction to Information Retrieval

We first review the distinction between structured and unstructured data (see Section 13.1) to see how information retrieval differs from structured data management. Consider a relation (or table) called HOUSES with the attributes:

HOUSES(Lot#, Address, Square\_footage, Listed\_price)

This is an example of *structured data*. We can compare this relation with home-buying contract documents, which are examples of *unstructured data*. These types of documents can vary from city to city, and even county to county, within a given state in the United States. Typically, a contract document in a particular state will have a standard list of clauses described in paragraphs within sections of the document, with some predetermined (fixed) text and some variable areas whose content is to be supplied by the specific buyer and seller. Other variable information would include interest rate for financing, down-payment amount, closing dates, and so on. The documents could also include pictures taken during a home inspection. The information content in such documents can be considered *unstructured data* that can be stored in a variety of possible arrangements and formats. By **unstructured information**, we generally mean information that does not have a well-defined formal model and corresponding formal language for representation and reasoning, but rather is based on understanding of natural language.

With the advent of the World Wide Web (or Web, for short), the volume of unstructured information stored in messages and documents that contain textual and multimedia information has exploded. These documents are stored in a variety of standard formats, including HTML, XML (see Chapter 13), and several audio and video formatting standards. Information retrieval deals with the problems of storing, indexing, and retrieving (searching) such information to satisfy the needs of users. The problems that IR deals with are exacerbated by the fact that the number of Web pages and the number of social interaction events is already in the billions and is growing at a phenomenal rate. All forms of unstructured data described above are being added at the rates of millions per day, expanding the searchable space on the Web at rapidly increasing rates.

Historically, **information retrieval** is “the discipline that deals with the structure, analysis, organization, storage, searching, and retrieval of information” as defined



by Gerald Salton, an IR pioneer.<sup>2</sup> We can enhance the definition slightly to say that it applies in the context of unstructured documents to satisfy a user's information needs. This field has existed even longer than the database field and was originally concerned with retrieval of cataloged information in libraries based on titles, authors, topics, and keywords. In academic programs, the field of IR has long been a part of Library and Information Science programs. Information in the context of IR does not require machine-understandable structures, such as in relational database systems. Examples of such information include written texts, abstracts, documents, books, Web pages, e-mails, instant messages, and collections from digital libraries. Therefore, all loosely represented (unstructured) or semistructured information is also part of the IR discipline.

We introduced XML modeling and retrieval in Chapter 13 and discussed advanced data types, including spatial, temporal, and multimedia data, in Chapter 26. RDBMS vendors are providing modules to support many of these data types, as well as XML data, in the newer versions of their products. These newer versions are sometimes referred to as *extended RDBMSs*, or *object-relational database management systems* (ORDBMSs; see Chapter 12). The challenge of dealing with unstructured data is largely an information retrieval problem, although database researchers have been applying database indexing and search techniques to some of these problems.

IR systems go beyond database systems in that they do not limit the user to a specific query language, nor do they expect the user to know the structure (schema) or content of a particular database. IR systems use a user's information need expressed as a **free-form search request** (sometimes called a **keyword search query**, or just **query**) for interpretation by the system. Whereas the IR field historically dealt with cataloging, processing, and accessing text in the form of documents for decades, in today's world the use of Web search engines is becoming the dominant way to find information. The traditional problems of text indexing and making collections of documents searchable have been transformed by making the Web itself into a quickly accessible repository of human knowledge or a virtual digital library.

An IR system can be characterized at different levels: by types of *users*, types of *data*, and the types of the *information need*, along with the size and scale of the information repository it addresses. Different IR systems are designed to address specific problems that require a combination of different characteristics. These characteristics can be briefly described as follows:

**Types of Users.** Users can greatly vary in their abilities to interact with computational systems. This ability depends on a multitude of factors, such as education, culture, and past exposure to computational environments. The user may be an *expert user* (for example, a curator or a librarian) who is searching for specific information that is clear in his/her mind, understands the scope and the structure of the available repository, and forms relevant queries for the task, or a *layperson user* with a generic information need. The latter cannot create highly relevant queries for search (for example, students trying to find information about a new topic, researchers trying to assimilate different points of

---

<sup>2</sup>See Salton's 1968 book entitled *Automatic Information Organization and Retrieval*.

view about a historical issue, a scientist verifying a claim by another scientist, or a person trying to shop for clothing). Designing systems suitable for different types of users is an important topic of IR that is typically studied in a field known as Human-Computer Information Retrieval.

**Types of Data.** Search systems can be tailored to specific types of data. For example, the problem of retrieving information about a specific topic may be handled more efficiently by customized search systems that are built to collect and retrieve only information related to that specific topic. The information repository could be hierarchically organized based on a concept or topic hierarchy. These topical *domain-specific* or *vertical IR systems* are not as large as or as diverse as the generic World Wide Web, which contains information on all kinds of topics. Given that these domain-specific collections exist and may have been acquired through a specific process, they can be exploited much more efficiently by a specialized system. Types of data can have different dimensions, such as *velocity*, *variety*, *volume*, and *veracity*. We discussed these in Section 25.1.

**Types of Information Need.** In the context of Web search, users' information needs may be defined as navigational, informational, or transactional.<sup>3</sup> **Navigational search** refers to finding a particular piece of information (such as the Georgia Tech University Web site) that a user needs quickly. The purpose of **informational search** is to find current information about a topic (such as research activities in the college of computing at Georgia Tech—this is the classic IR system task). The goal of **transactional search** is to reach a site where further interaction happens resulting in some transactional event (such as joining a social network, shopping for products, making online reservations, accessing databases, and so on).

**Levels of Scale.** In the words of Nobel Laureate Herbert Simon,

*“What information consumes is rather obvious: it consumes the attention of its recipients. Hence a wealth of information creates a poverty of attention, and a need to allocate that attention efficiently among the overabundance of information sources that might consume it.”*<sup>4</sup>

This overabundance of information sources in effect creates a high noise-to-signal ratio in IR systems. Especially on the Web, where billions of pages are indexed, IR interfaces are built with efficient scalable algorithms for distributed searching, indexing, caching, merging, and fault tolerance. IR search engines can be limited in level to more specific collections of documents. **Enterprise search systems** offer IR solutions for searching different entities in an enterprise's **intranet**, which consists of the network of computers within that enterprise. The searchable entities include e-mails, corporate documents, manuals, charts, and presentations, as well as reports related to people, meetings, and projects. Enterprise search systems still typically deal with hundreds of millions of entities in large global enterprises. On a smaller scale, there are personal information systems such as those on desktops and laptops, called

<sup>3</sup>See Broder (2002) for details.

<sup>4</sup>From Herbert A. Simon (1971), “Designing Organizations for an Information-Rich World.”

**desktop search engines** (for example, Google Desktop, OS X Spotlight), for retrieving files, folders, and different kinds of entities stored on the computer. There are other systems that use peer-to-peer technology, such as the BitTorrent protocol, which allows sharing of music in the form of audio files, as well as specialized search engines for audio, such as Lycos and Yahoo! audio search.

### 27.1.2 Databases and IR Systems: A Comparison

Within the computer science discipline, databases and IR systems are closely related fields. Databases deal with structured information retrieval through well-defined formal languages for representation and manipulation based on the theoretically founded data models. Efficient algorithms have been developed for operators that allow rapid execution of complex queries. IR, on the other hand, deals with unstructured search with possibly vague query or search semantics and without a well-defined logical schematic representation. Some of the key differences between databases and IR systems are listed in Table 27.1.

Whereas databases have fixed schemas defined in some data model such as the relational model, an IR system has no fixed data model; it views data or documents according to some scheme, such as the vector space model, to aid in query processing (see Section 27.2). Databases using the relational model employ SQL for queries and transactions. The queries are mapped into relational algebra operations and search algorithms (see Chapter 19) and return a new relation (table) as the query result, providing an exact answer to the query for the current state of the database. In IR systems, there is no fixed language for defining the structure (schema) of the document or for operating on the document—queries tend to be a set of query terms (keywords) or a free-form natural language phrase. An IR query result is a list of document id's, or some pieces of text or multimedia objects (images, videos, and so on), or a list of links to Web pages.

The result of a database query is an exact answer; if no matching records (tuples) are found in the relation, the result is empty (null). On the other hand, the answer

**Table 27.1** A Comparison of Databases and IR Systems

Databases	IR Systems
<ul style="list-style-type: none"> <li>■ Structured data</li> <li>■ Schema driven</li> <li>■ Relational (or object, hierarchical, and network) model is predominant</li> <li>■ Structured query model</li> <li>■ Rich metadata operations</li> <li>■ Query returns data</li> <li>■ Results are based on exact matching (always correct)</li> </ul>	<ul style="list-style-type: none"> <li>■ Unstructured data</li> <li>■ No fixed schema; various data models (e.g., vector space model)</li> <li>■ Free-form query models</li> <li>■ Rich data operations</li> <li>■ Search request returns list or pointers to documents</li> <li>■ Results are based on approximate matching and measures of effectiveness (may be imprecise and ranked)</li> </ul>

to a user request in an IR query represents the IR system's best attempt at retrieving the information most relevant to that query. Whereas database systems maintain a large amount of metadata and allow their use in query optimization, the operations in IR systems rely on the data values themselves and their occurrence frequencies. Complex statistical analysis is sometimes performed to determine the *relevance* of each document or parts of a document to the user request.

### 27.1.3 A Brief History of IR

Information retrieval has been a common task since the times of ancient civilizations, which devised ways to organize, store, and catalog documents and records. Media such as papyrus scrolls and stone tablets were used to record documented information in ancient times. These efforts allowed knowledge to be retained and transferred among generations. With the emergence of public libraries and the printing press, large-scale methods for producing, collecting, archiving, and distributing documents and books evolved. As computers and automatic storage systems emerged, the need to apply these methods to computerized systems arose. Several techniques emerged in the 1950s, such as the seminal work of H. P. Luhn,<sup>5</sup> who proposed using words and their frequency counts as indexing units for documents, and using measures of word overlap between queries and documents as the retrieval criterion. It was soon realized that storing large amounts of text was not difficult. The harder task was to search for and retrieve that information selectively for users with specific information needs. Methods that explored word distribution statistics gave rise to the choice of keywords based on their distribution properties<sup>6</sup> and also led to keyword-based weighting schemes.

The earlier experiments with document retrieval systems such as SMART<sup>7</sup> in the 1960s adopted the *inverted file organization* based on keywords and their weights as the method of indexing (see Section 17.6.4 on inverted indexing). Serial (or sequential) organization proved inadequate if queries required fast, near real-time response times. Proper organization of these files became an important area of study; document classification and clustering schemes ensued. The scale of retrieval experiments remained a challenge due to lack of availability of large text collections. This soon changed with the World Wide Web. Also, the Text Retrieval Conference (TREC) was launched by NIST (National Institute of Standards and Technology) in 1992 as a part of the TIPSTER program<sup>8</sup> with the goal of providing a platform for evaluating information retrieval methodologies and facilitating technology transfer to develop IR products.

A **search engine** is a practical application of information retrieval to large-scale document collections. With significant advances in computers and communications technologies, people today have interactive access to enormous amounts of user-generated

---

<sup>5</sup>See Luhn (1957) "A statistical approach to mechanized encoding and searching of literary information."

<sup>6</sup>See Salton, Yang, and Yu (1975).

<sup>7</sup>For details, see Buckley et al. (1993).

<sup>8</sup>For details, see Harman (1992).

distributed content on the Web. This has spurred the rapid growth in search engine technology, where search engines are trying to discover different kinds of real-time content found on the Web. The part of a search engine responsible for discovering, analyzing, and indexing these new documents is known as a **crawler**. Other types of search engines exist for specific domains of knowledge. For example, the biomedical literature search database was started in the 1970s and is now supported by the PubMed search engine,<sup>9</sup> which gives access to over 24 million abstracts.

Although continuous progress is being made to tailor search results to the needs of an end user, the challenge remains in providing high-quality, pertinent, and timely information that is precisely aligned to the information needs of individual users.

### 27.1.4 Modes of Interaction in IR Systems

In the beginning of Section 27.1, we defined *information retrieval* as the process of retrieving documents from a collection in response to a query (or a search request) by a user. Typically the collection is made up of documents containing unstructured data. Other kinds of documents include images, audio recordings, video strips, and maps. Data may be scattered nonuniformly in these documents with no definitive structure. A **query** is a set of **terms** (also referred to as **keywords**) used by the searcher to specify an information need (for example, the terms *databases* and *operating systems* may be regarded as a query to a computer science bibliographic database). An informational request or a search query may also be a natural language phrase or a question (for example, “What is the currency of China?” or “Find Italian restaurants in Sarasota, Florida.”).

There are two main modes of interaction with IR systems—retrieval and browsing—which, although similar in goal, are accomplished through different interaction tasks. **Retrieval** is concerned with the extraction of relevant information from a repository of documents through an IR query, whereas **browsing** signifies the exploratory activity of a user visiting or navigating through similar or related documents based on the user’s assessment of relevance. During browsing, a user’s information need may not be defined *a priori* and is flexible. Consider the following browsing scenario: A user specifies ‘Atlanta’ as a keyword. The information retrieval system retrieves links to relevant result documents containing various aspects of Atlanta for the user. The user comes across the term ‘Georgia Tech’ in one of the returned documents and uses some access technique (such as clicking on the phrase ‘Georgia Tech’ in a document that has a built-in link) and visits documents about Georgia Tech in the same or a different Web site (repository). There the user finds an entry for ‘Athletics’ that leads the user to information about various athletic programs at Georgia Tech. Eventually, the user ends his search at the Fall schedule for the Yellow Jackets football team, which he finds to be of great interest. This user activity is known as browsing. **Hyperlinks** are used to interconnect Web pages and are mainly used for browsing. **Anchor texts** are text phrases within documents used to label hyperlinks and are very relevant to browsing.

<sup>9</sup>See [www.ncbi.nlm.nih.gov/pubmed/](http://www.ncbi.nlm.nih.gov/pubmed/)

**Web search** combines both aspects—browsing and retrieval—and is one of the main applications of information retrieval today. Web pages are analogous to documents. Web search engines maintain an indexed repository of Web pages, usually using the technique of inverted indexing (see Section 27.5). They retrieve the most relevant Web pages for the user in response to the user’s search request with a possible ranking in descending order of relevance. The **rank of a Web page** in a retrieved set is the measure of its relevance to the query that generated the result set.

### 27.1.5 Generic IR Pipeline

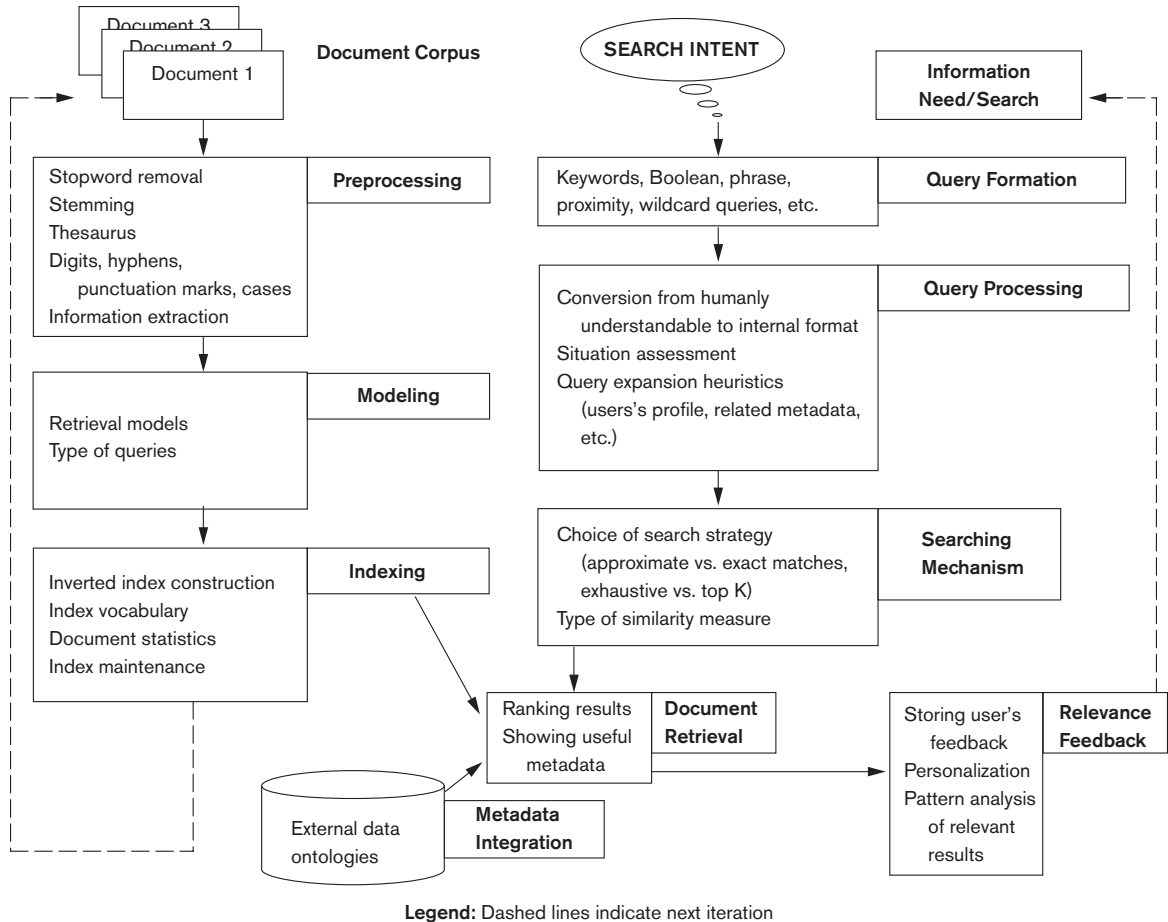
As we mentioned earlier, documents are made up of unstructured natural language text composed of character strings from English and other languages. Common examples of documents include newswire services (such as AP or Reuters), corporate manuals and reports, government notices, Web page articles, blogs, tweets, books, and journal papers. There are two main approaches to IR: statistical and semantic.

In a **statistical approach**, documents are analyzed and broken down into chunks of text (words, phrases, or  $n$ -grams, which are all subsequences of length  $n$  characters in a text or document) and each word or phrase is counted, weighted, and measured for relevance or importance. These words and their properties are then compared with the query terms for potential degree of match to produce a ranked list of resulting documents that contain the words. Statistical approaches are further classified based on the method employed. The three main statistical approaches are Boolean, vector space, and probabilistic (see Section 27.2).

**Semantic approaches** to IR use knowledge-based techniques of retrieval that broadly rely on the syntactic, lexical, sentential, discourse-based, and pragmatic levels of knowledge understanding. In practice, semantic approaches also apply some form of statistical analysis to improve the retrieval process.

Figure 27.1 shows the various stages involved in an IR processing system. The steps shown on the left in Figure 27.1 are typically offline processes, which prepare a set of documents for efficient retrieval; these are document preprocessing, document modeling, and indexing. The right side of Figure 27.1 deals with the process of a user interacting with the IR system either during a querying, browsing, or searching. It shows the steps involved; namely, query formation, query processing, searching mechanism, document retrieval, and relevance feedback. In each box, we highlight the important concepts and issues. The rest of this chapter describes some of the concepts involved in the various tasks within the IR process shown in Figure 27.1.

Figure 27.2 shows a simplified IR processing pipeline. In order to perform retrieval on documents, the documents are first represented in a form suitable for retrieval. The significant terms and their properties are extracted from the documents and are represented in a document index where the words/terms and their properties are stored in a matrix that contains each individual document in a row and each row contains the references to the words contained in those documents. This index is then converted into an inverted index (see Figure 27.4) of a word/term versus document matrix. Given the query words, the documents containing these words—



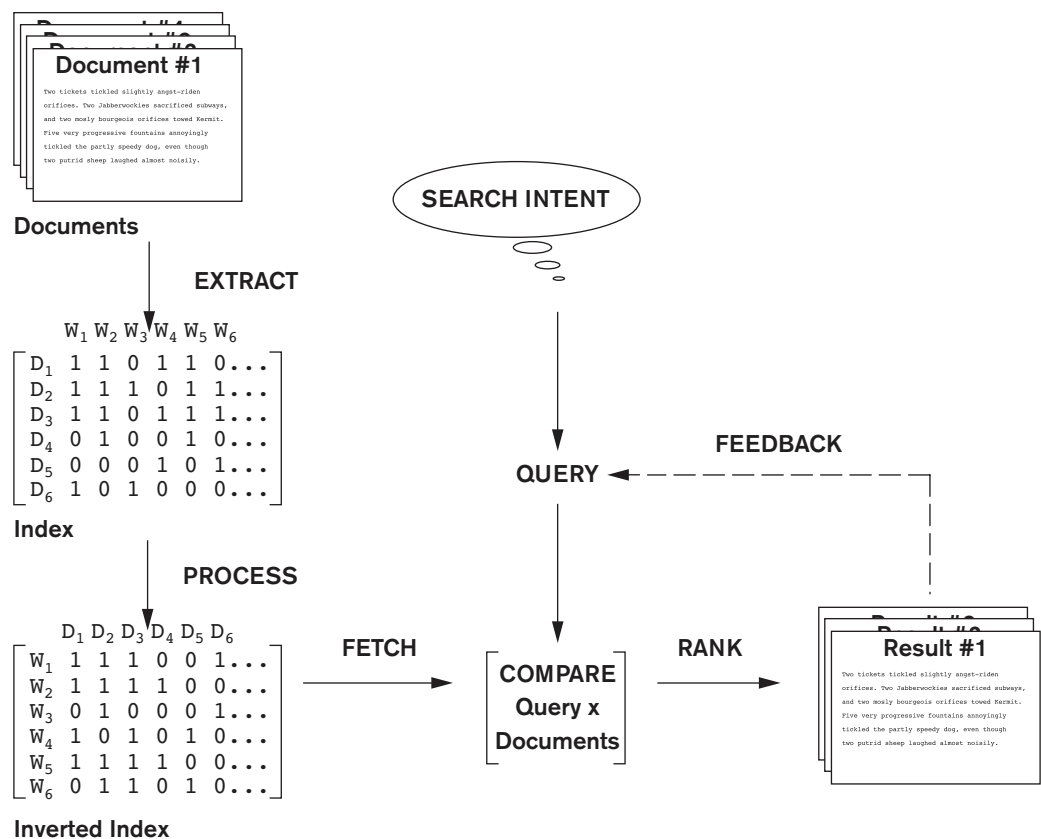
**Figure 27.1**  
Generic IR framework.

and the document properties, such as date of creation, author, and type of document—are fetched from the inverted index and compared with the query. This comparison results in a ranked list shown to the user. The user can then provide feedback on the results that triggers implicit or explicit query modification and expansion to fetch results that are more relevant for the user. Most IR systems allow for an interactive search in which the query and the results are successively refined.

## 27.2 Retrieval Models

In this section, we briefly describe the important models of IR. These are the three main statistical models—Boolean, vector space, and probabilistic—and the semantic model.





**Figure 27.2**  
Simplified IR process pipeline.

### 27.2.1 Boolean Model

In this model, documents are represented as a set of *terms*. Queries are formulated as a combination of terms using the standard Boolean logic set-theoretic operators such as AND, OR and NOT. Retrieval and relevance are considered as binary concepts in this model, so the retrieved elements are an “exact match” retrieval of relevant documents. There is no notion of ranking of resulting documents. All retrieved documents are considered equally important—a major simplification that does not consider frequencies of document terms or their proximity to other terms compared against the query terms.

Boolean retrieval models lack sophisticated ranking algorithms and are among the earliest and simplest information retrieval models. These models make it easy to associate metadata information and write queries that match the contents of the documents as well as other properties of documents, such as date of creation, author, and type of document.



### 27.2.2 Vector Space Model

The vector space model provides a framework in which term weighting, ranking of retrieved documents, and determining the relevance of feedback are possible. Using individual terms as dimensions, each document is represented by an  $n$ -dimensional vector of values. The values themselves may be a Boolean value to represent the existence or absence of the term in that document; alternately, they may be a number representative of the weight or frequency in the document. **Features** are a subset of the terms in a *set of documents* that are deemed most relevant to an IR search for this particular set of documents. The process of selecting these important terms (features) and their properties as a sparse (limited) list out of the very large number of available terms (the vocabulary can contain hundreds of thousands of terms) is independent of the model specification. The query is also specified as a terms vector (vector of features), and this is compared to the document vectors for similarity/relevance assessment.

The similarity assessment function that compares two vectors is not inherent to the model—different similarity functions can be used. However, the cosine of the angle between the query and document vector is a commonly used function for similarity assessment. As the angle between the vectors decreases, the cosine of the angle approaches one, meaning that the similarity of the query with a document vector increases. Terms (features) are weighted proportional to their frequency counts to reflect the importance of terms in the calculation of relevance measure. This is different from the Boolean model, which does not take into account the frequency of words in the document for relevance match.

In the vector model, the *document term weight*  $w_{ij}$  (for term  $i$  in document  $j$ ) is represented based on some variation of the TF (term frequency) or TF-IDF (term frequency–inverse document frequency) scheme (as we will describe below). **TF-IDF** is a statistical weight measure that is used to evaluate the importance of a document word in a collection of documents. The following formula is typically used:

$$\text{cosine}(d_j, q) = \frac{\langle d_j \times q \rangle}{\|d_j\| \times \|q\|} = \frac{\sum_{i=1}^{|V|} w_{ij} \times w_{iq}}{\sqrt{\sum_{i=1}^{|V|} w_{ij}^2} \times \sqrt{\sum_{i=1}^{|V|} w_{iq}^2}}$$

In the formula given above, we use the following symbols:

- $d_j$  is the document vector for document  $j$ .
- $q$  is the query vector.
- $w_{ij}$  is the weight of term  $i$  in document  $j$ .
- $w_{iq}$  is the weight of term  $i$  in query vector  $q$ .
- $|V|$  is the number of dimensions in the vector that is the total number of important keywords (or features).

TF-IDF uses the product of normalized frequency of a term  $i$  ( $TF_{ij}$ ) in document  $D_j$  and the inverse document frequency of the term  $i$  ( $IDF_i$ ) to weight a term in a

document. The idea is that terms that capture the essence of a document occur frequently in the document (that is, their TF is high), but if such a term were to be a good term that discriminates the document from others, it must occur in only a few documents in the general population (that is, its IDF should be high as well).

IDF values can be easily computed for a fixed collection of documents. In case of Web search engines, taking a representative sample of documents approximates IDF computation. The following formulas can be used:

$$TF_{ij} = f_{ij} / \sum_{i=1 \text{ to } |V|} f_{ij}$$

$$IDF_i = \log(N / n_i)$$

In these formulas, the meaning of the symbols is:

- $TF_{ij}$  is the normalized term frequency of term  $i$  in document  $D_j$ .
- $f_{ij}$  is the number of occurrences of term  $i$  in document  $D_j$ .
- $IDF_i$  is the inverse document frequency weight for term  $i$ .
- $N$  is the number of documents in the collection.
- $n_i$  is the number of documents in which term  $i$  occurs.

Note that if a term  $i$  occurs in all documents, then  $n_i = N$  and hence  $IDF_i = \log(1)$  becomes zero, nullifying its importance and creating a situation where division by zero can occur. The weight of term  $i$  in document  $j$ ,  $w_{ij}$ , is computed based on its TF-IDF value in some techniques. To prevent division by zero, it is common to add a 1 to the denominator in the formulae such as the cosine formula above.

Sometimes, the relevance of the document with respect to a query ( $\text{rel}(D_j, Q)$ ) is directly measured as the sum of the TF-IDF values of the terms in the query  $Q$ :

$$\text{rel}(D_j, Q) = \sum_{i \in Q} TF_{ij} \times IDF_i$$

The normalization factor (similar to the denominator of the cosine formula) is incorporated into the TF-IDF formula itself, thereby measuring relevance of a document to the query by the computation of the dot product of the query and document vectors.

The Rocchio<sup>10</sup> algorithm is a well-known relevance feedback algorithm based on the vector space model that modifies the initial query vector and its weights in response to user-identified relevant documents. It expands the original query vector  $q$  to a new vector  $q_e$  as follows:

$$q_e = \alpha q + \frac{\beta}{|D_r|} \sum_{d_r \in D_r} d_r - \frac{\gamma}{|D_{nr}|} \sum_{d_{nr} \in D_{nr}} d_{nr},$$

---

<sup>10</sup>See Rocchio (1971).

Here,  $D_r$  stands for document-relevant ( $D_r$ ) and  $D_{nr}$  stands for document-nonrelevant ( $D_{nr}$ ); these terms represent relevant and nonrelevant document sets, respectively. Terms from relevant and nonrelevant documents get added to the original query vector with positive and negative weights, respectively, to create the modified query vector.  $\alpha$ ,  $\beta$ , and  $\gamma$  are parameters of the equation. The summation over  $d_r$  represents summation over all relevant terms of document  $d_r$ . Similarly, summation over  $d_{nr}$  represents summation over all nonrelevant terms of document  $d_{nr}$ . The values of these parameters determine how the feedback affects the original query, and these may be determined after a number of trial-and-error experiments.

### 27.2.3 Probabilistic Model

The similarity measures in the vector space model are somewhat ad hoc. For example, the model assumes that those documents closer to the query in cosine space are more relevant to the query vector. In the probabilistic model, a more concrete and definitive approach is taken: ranking documents by their estimated probability of relevance with respect to the query and the document. This is the basis of the *probability ranking principle* developed by Robertson.<sup>11</sup>

In the probabilistic framework, the IR system must decide whether the documents belong to the **relevant set** or the **nonrelevant set** for a query. To make this decision, it is assumed that a predefined relevant set and nonrelevant set exist for the query, and the task is to calculate the probability that the document belongs to the relevant set and compare that with the probability that the document belongs to the nonrelevant set.

Given the document representation  $D$  of a document, estimating the relevance  $R$  and nonrelevance  $NR$  of that document involves computation of conditional probability  $P(R|D)$  and  $P(NR|D)$ . These conditional probabilities can be calculated using Bayes' rule:<sup>12</sup>

$$\begin{aligned} P(R|D) &= P(D|R) \times P(R)/P(D) \\ P(NR|D) &= P(D|NR) \times P(NR)/P(D) \end{aligned}$$

A document  $D$  is classified as relevant if  $P(R|D) > P(NR|D)$ . Discarding the constant  $P(D)$ , this is equivalent to saying that a document is relevant if:

$$P(D|R) \times P(R) > P(D|NR) \times P(NR)$$

The likelihood ratio  $P(D|R)/P(D|NR)$  is used as a score to determine the likelihood of the document with representation  $D$  belonging to the relevant set.

The *term independence* or *naïve Bayes* assumption is used to estimate  $P(D|R)$  using computation of  $P(t_i|R)$  for term  $t_i$ . The likelihood ratios  $P(D|R)/P(D|NR)$  of documents are used as a proxy for ranking based on the assumption that highly ranked documents will have a high likelihood of belonging to the relevant set.<sup>13</sup>

<sup>11</sup>For a description of the Cheshire II system, see Robertson (1997).

<sup>12</sup>Bayes' theorem is a standard technique for measuring likelihood; see Howson and Urbach (1993), for example.

<sup>13</sup>Readers should refer to Croft et al. (2009) pages 246–247 for a detailed description.

With some reasonable assumptions and estimates about the probabilistic model along with extensions for incorporating query term weights and document term weights in the model, a probabilistic ranking algorithm called **BM25** (Best Match 25) is quite popular. This weighting scheme has evolved from several versions of the **Okapi**<sup>14</sup> system.

The Okapi weight for document  $d_j$  and query  $q$  is computed by the formula below. Additional notations are as follows:

- $t_i$  is a term.
- $f_{ij}$  is the raw frequency count of term  $t_i$  in document  $d_j$ .
- $f_{iq}$  is the raw frequency count of term  $t_i$  in query  $q$ .
- $N$  is the total number of documents in the collection.
- $df_i$  is the number of documents that contain the term  $t_i$ .
- $dl_j$  is the document length (in bytes) of  $d_j$ .
- $avdl$  is the average document length of the collection.

The Okapi relevance score of a document  $d_j$  for a query  $q$  is given by the equation below, where  $k_1$  (between 1.0–2.0),  $b$  (usually 0.75), and  $k_2$  (between 1–1,000) are parameters:

$$\text{okapi}(d_j, q) = \sum_{t_i \in q, d_j} \ln \frac{N - df_i + 0.5}{df_i + 0.5} \times \frac{(k_1 + 1)f_{ij}}{k_1 \left( 1 - b + b \frac{dl_j}{avdl} \right) + f_{ij}} \times \frac{(k_2 + 1)f_{iq}}{k_2 + f_{iq}}$$

### 27.2.4 Semantic Model

However sophisticated the above statistical models become, they can miss many relevant documents because those models do not capture the complete meaning or information need conveyed by a user's query. In semantic models, the process of matching documents to a given query is based on concept level and semantic matching instead of index term (keyword) matching. This allows retrieval of relevant documents that share meaningful associations with other documents in the query result, even when these associations are not inherently observed or statistically captured.

Semantic approaches include different levels of analysis, such as morphological, syntactic, and semantic analysis, to retrieve documents more effectively. In **morphological analysis**, roots and affixes are analyzed to determine the parts of speech (nouns, verbs, adjectives, and so on) of the words. Following morphological analysis, **syntactic analysis** follows to parse and analyze complete phrases in documents. Finally, the semantic methods have to resolve word ambiguities and/or generate relevant synonyms based on the **semantic relationships** among levels of structural entities in documents (words, paragraphs, pages, or entire documents).

<sup>14</sup>City University of London Okapi System by Robertson, Walker, and Hancock-Beaulieu (1995).

The development of a sophisticated semantic system requires complex knowledge bases of semantic information as well as retrieval heuristics. These systems often require techniques from artificial intelligence and expert systems. Knowledge bases like Cyc<sup>15</sup> and WordNet<sup>16</sup> have been developed for use in *knowledge-based IR systems* based on semantic models. The Cyc knowledge base, for example, is a representation of a vast quantity of commonsense knowledge. It presently contains 15.94 million assertions, 498,271 atomic concepts, and 441,159 nonatomic derived concepts for reasoning about the objects and events of everyday life. WordNet is an extensive thesaurus (over 117,000 concepts) that is very popular and is used by many systems and is under continuous development (see Section 27.4.3).

## 27.3 Types of Queries in IR Systems

Different keywords are associated with the document set during the process of indexing. These keywords generally consist of words, phrases, and other characterizations of documents such as date created, author names, and type of document. They are used by an IR system to build an inverted index (see Section 27.5), which is then consulted during the search. The queries formulated by users are compared to the set of index keywords. Most IR systems also allow the use of Boolean and other operators to build a complex query. The query language with these operators enriches the expressiveness of a user's information need.

### 27.3.1 Keyword Queries

Keyword-based queries are the simplest and most commonly used forms of IR queries: the user just enters keyword combinations to retrieve documents. The query keyword terms are implicitly connected by a logical AND operator. A query such as 'database concepts' retrieves documents that contain both the words 'database' and 'concepts' at the top of the retrieved results. In addition, most systems also retrieve documents that contain only 'database' or only 'concepts' in their text. Some systems remove most commonly occurring words (such as *a*, *the*, *of*, and so on, called **stopwords**) as a preprocessing step before sending the filtered query keywords to the IR engine. Most IR systems do not pay attention to the ordering of these words in the query. All retrieval models provide support for keyword queries.

### 27.3.2 Boolean Queries

Some IR systems allow using the AND, OR, NOT, ( ), +, and – Boolean operators in combinations of keyword formulations. AND requires that both terms be found. OR lets either term be found. NOT means any record containing the second term will be excluded. '( )' means the Boolean operators can be nested using parentheses. '+' is equivalent to AND, requiring the term; the '+' should be placed directly in front

---

<sup>15</sup>See Lenat (1995).

<sup>16</sup>See Miller (1990) for a detailed description of WordNet.

of the search term. ‘-’ is equivalent to AND NOT and means to exclude the term; the ‘-’ should be placed directly in front of the search term not wanted. Complex Boolean queries can be built out of these operators and their combinations, and they are evaluated according to the classical rules of Boolean algebra. No ranking is possible, because a document either satisfies such a query (is “relevant”) or does not satisfy it (is “nonrelevant”). A document is retrieved for a Boolean query if the query is logically true as an exact match in the document. Users generally do not use combinations of these complex Boolean operators, and IR systems support a restricted version of these set operators. Boolean retrieval models can directly support different Boolean operator implementations for these kinds of queries.

### 27.3.3 Phrase Queries

When documents are represented using an inverted keyword index for searching, the relative order of the terms in the document is lost. In order to perform exact phrase retrieval, these phrases should be encoded in the inverted index or implemented differently (with relative positions of word occurrences in documents). A phrase query consists of a sequence of words that makes up a phrase. The phrase is generally enclosed within double quotes. Each retrieved document must contain at least one instance of the exact phrase. Phrase searching is a more restricted and specific version of proximity searching that we mention below. For example, a phrase searching query could be ‘conceptual database design’. If phrases are indexed by the retrieval model, any retrieval model can be used for these query types. A phrase thesaurus may also be used in semantic models for fast dictionary searching of phrases.

### 27.3.4 Proximity Queries

Proximity search refers to a search that accounts for how close within a record multiple terms should be to each other. The most commonly used proximity search option is a phrase search that requires terms to be in the exact order. Other proximity operators can specify how close terms should be to each other. Some will also specify the order of the search terms. Each search engine can define proximity operators differently, and the search engines use various operator names such as NEAR, ADJ(adjacent), or AFTER. In some cases, a sequence of single words is given, together with a maximum allowed distance between them. Vector space models that also maintain information about positions and offsets of tokens (words) have robust implementations for this query type. However, providing support for complex proximity operators becomes computationally expensive because it requires the time-consuming preprocessing of documents and is thus suitable for smaller document collections rather than for the Web.

### 27.3.5 Wildcard Queries

Wildcard searching is generally meant to support regular expressions and pattern matching-based searching in text. In IR systems, certain kinds of wildcard search support may be implemented—usually words with any trailing characters (for example, ‘data\*’ would retrieve *data*, *database*, *datapoint*, *dataset*, and so on). Providing full support

for wildcard searches in Web search engines involves preprocessing overhead and is not generally implemented by many Web search engines today.<sup>17</sup> Retrieval models do not directly provide support for this query type. Lucene<sup>18</sup> provides support for certain types of wildcard queries. The query parser in Lucene computes a large Boolean query combining all combinations and expansions of words from the index.

### 27.3.6 Natural Language Queries

There are a few natural language search engines that aim to understand the structure and meaning of queries written in natural language text, generally as a question or narrative. This is an active area of research that employs techniques like shallow semantic parsing of text, or query reformulations based on natural language understanding. The system tries to formulate answers for such queries from retrieved results. Some search systems are starting to provide natural language interfaces to provide answers to specific types of questions, such as definition and factoid questions, which ask for definitions of technical terms or common facts that can be retrieved from specialized databases. Such questions are usually easier to answer because there are strong linguistic patterns giving clues to specific types of sentences—for example, ‘defined as’ or ‘refers to’. Semantic models can provide support for this query type.

## 27.4 Text Preprocessing

In this section, we review the commonly used text preprocessing techniques that are part of the text processing task in Figure 27.1.

### 27.4.1 Stopword Removal

**Stopwords** are very commonly used words in a language that play a major role in the formation of a sentence but that seldom contribute to the meaning of that sentence. Words that are expected to occur in 80% or more of the documents in a collection are typically referred to as *stopwords*, and they are rendered potentially useless. Because of the commonness and function of these words, they do not contribute much to the relevance of a document for a query search. Examples include words such as *the, of, to, a, and, in, said, for, that, was, on, he, is, with, at, by, and it*. These words are presented here with decreasing frequency of occurrence from a large corpus of documents called **AP89**.<sup>19</sup> The first six of these words account for 20% of all words in the listing, and the most frequent 50 words account for 40% of all text.

Removal of stopwords from a document must be performed before indexing. Articles, prepositions, conjunctions, and some pronouns are generally classified as stopwords. Queries must also be preprocessed for stopwords removal before the actual retrieval process. Removal of stopwords results in elimination of possible spurious

<sup>17</sup>See [http://www.livinginternet.com/w/wu\\_expert\\_wild.htm](http://www.livinginternet.com/w/wu_expert_wild.htm) for further details.

<sup>18</sup><http://lucene.apache.org/>

<sup>19</sup>For details, see Croft et al. (2009), pages 75–90.



indexes, thereby reducing the size of an index structure by about 40% or more. However, doing so could impact the recall if the stopword is an integral part of a query (for example, a search for the phrase ‘To be or not to be’, where removal of stopwords makes the query inappropriate, as all the words in the phrase are stopwords). Many search engines do not employ query stopword removal for this reason.

### 27.4.2 Stemming

A **stem** of a word is defined as the word obtained after trimming the suffix and prefix of an original word. For example, ‘comput’ is the stem word for *computer*, *computing*, *computable*, and *computation*. These suffixes and prefixes are very common in the English language for supporting the notion of verbs, tenses, and plural forms. **Stemming** reduces the different forms of the word formed by inflection (due to plurals or tenses) and derivation to a common stem.

A stemming algorithm can be applied to reduce any word to its stem. In English, the most famous stemming algorithm is Martin Porter’s stemming algorithm. The Porter stemmer<sup>20</sup> is a simplified version of Lovin’s technique that uses a reduced set of about 60 rules (from 260 suffix patterns in Lovin’s technique) and organizes them into sets; conflicts within one subset of rules are resolved before going on to the next. Using stemming for preprocessing data results in a decrease in the size of the indexing structure and an increase in recall, possibly at the cost of precision.

### 27.4.3 Utilizing a Thesaurus

A **thesaurus** comprises a precompiled list of important concepts and the main word that describes each concept for a particular domain of knowledge. For each concept in this list, a set of synonyms and related words is also compiled.<sup>21</sup> Thus, a synonym can be converted to its matching concept during preprocessing. This preprocessing step assists in providing a standard vocabulary for indexing and searching. Usage of a thesaurus, also known as a *collection of synonyms*, has a substantial impact on the recall of information systems. This process can be complicated because many words have different meanings in different contexts.

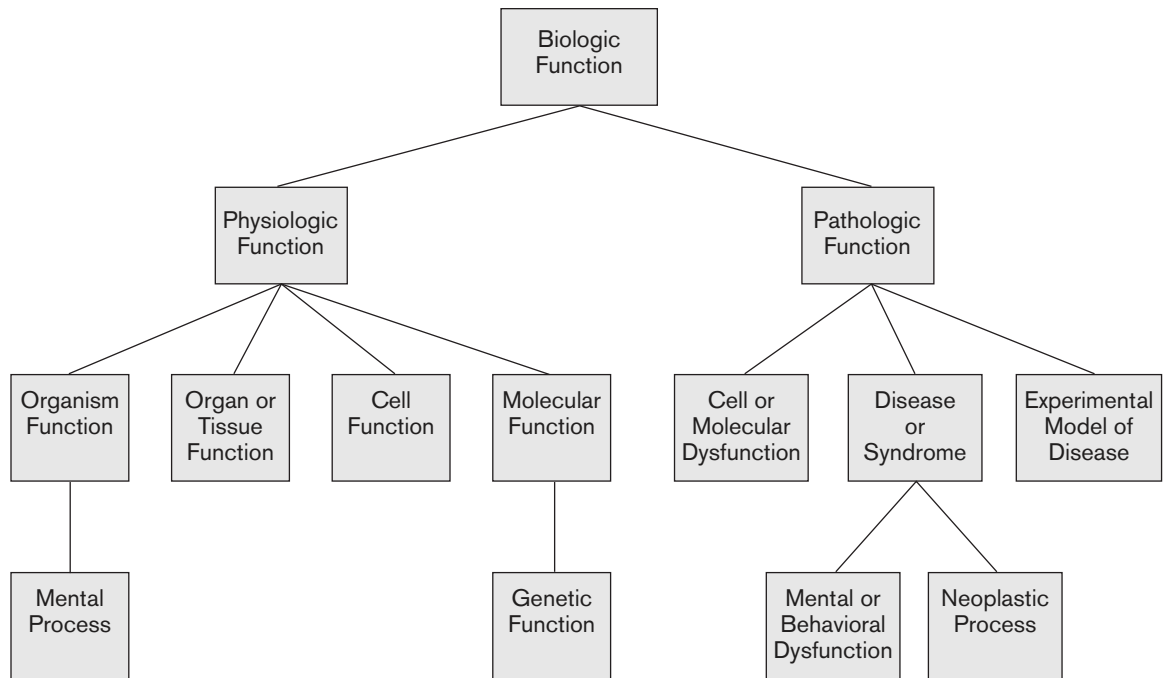
**UMLS**<sup>22</sup> is a large biomedical thesaurus of millions of concepts (called the *meta-thesaurus*) and a semantic network of meta concepts and relationships that organize the metathesaurus (see Figure 27.3). The concepts are assigned labels from the semantic network. This thesaurus of concepts contains synonyms of medical terms, hierarchies of broader and narrower terms, and other relationships among words and concepts that make it a very extensive resource for information retrieval of documents in the medical domain. Figure 27.3 illustrates part of the UMLS Semantic Network.

---

<sup>20</sup>See Porter (1980).

<sup>21</sup>See Baeza-Yates and Ribeiro-Neto (1999).

<sup>22</sup>Unified Medical Language System from the National Library of Medicine.

**Figure 27.3**

A portion of the UMLS Semantic Network: “Biologic Function” Hierarchy.

Source: UMLS Reference Manual, National Library of Medicine.

**WordNet**<sup>23</sup> is a manually constructed thesaurus that groups words into strict synonym sets called *synsets*. These synsets are divided into noun, verb, adjective, and adverb categories. Within each category, these synsets are linked together by appropriate relationships such as class/subclass or “is-a” relationships for nouns.

WordNet is based on the idea of using a controlled vocabulary for indexing, thereby eliminating redundancies. It is also useful in providing assistance to users with locating terms for proper query formulation.

#### 27.4.4 Other Preprocessing Steps: Digits, Hyphens, Punctuation Marks, Cases

Digits, dates, phone numbers, e-mail addresses, URLs, and other standard types of text may or may not be removed during preprocessing. Web search engines, however, index them in order to use this type of information in the document metadata to improve precision and recall (see Section 27.6 for detailed definitions of *precision* and *recall*).

<sup>23</sup>See Fellbaum (1998) for a detailed description of WordNet.

Hyphens and punctuation marks may be handled in different ways. Either the entire phrase with the hyphens/punctuation marks may be used, or they may be eliminated. In some systems, the character representing the hyphen/punctuation mark may be removed, or may be replaced with a space. Different information retrieval systems follow different rules of processing. Handling hyphens automatically can be complex: it can either be done as a classification problem, or more commonly by some heuristic rules. For example, the `StandardTokenizer` in Lucene<sup>24</sup> treats the hyphen as a delimiter to break words—with the exception that if there is a number in the token, the words are not split (for example, words like AK-47, phone numbers, etc.). Many domain-specific terms like product catalogs, different versions of a product, and so on have hyphens in them. When search engines crawl the Web for indexing, it becomes difficult to automatically treat hyphens correctly; therefore, simpler strategies are devised to process hyphens.

Most information retrieval systems perform case-insensitive search, converting all the letters of the text to uppercase or lowercase. It is also worth noting that many of these text preprocessing steps are language specific, such as involving accents and diacritics and the idiosyncrasies that are associated with a particular language.

### 27.4.5 Information Extraction

**Information extraction** (IE) is a generic term used for extracting structured content from text. Text analytic tasks such as identifying noun phrases, facts, events, people, places, and relationships are examples of IE tasks. These tasks are also called *named entity recognition tasks* and use rule-based approaches with either a thesaurus, regular expressions and grammars, or probabilistic approaches. For IR and search applications, IE technologies are mostly used to identify named entities that involve text analysis, matching, and categorization for improving the relevance of search systems. Language technologies using part-of-speech tagging are applied to semantically annotate the documents with extracted features to aid search relevance.

## 27.5 Inverted Indexing

The simplest way to search for occurrences of query terms in text collections can be performed by sequentially scanning the text. This kind of online searching is only appropriate when text collections are small. Most information retrieval systems process the text collections to create indexes and operate upon the inverted index data structure (refer to the indexing task in Figure 27.1). An inverted index structure comprises vocabulary and document information. **Vocabulary** is a set of distinct query terms in the document set. Each term in a vocabulary set has an associated collection of information about the documents that contain the term, such as document id, occurrence count, and offsets within the document where the

---

<sup>24</sup>See further details on `StandardTokenizer` at <https://lucene.apache.org/>

term occurs. The simplest form of vocabulary terms consists of words or individual tokens of the documents. In some cases, these vocabulary terms also consist of phrases,  $n$ -grams, entities, links, names, dates, or manually assigned descriptor terms from documents and/or Web pages. For each term in the vocabulary, the corresponding document id's, occurrence locations of the term in each document, number of occurrences of the term in each document, and other relevant information may be stored in the document information section.

Weights are assigned to document terms to represent an estimate of the usefulness of the given term as a descriptor for distinguishing the given document from other documents in the same collection. A term may be a better descriptor of one document than of another by the weighting process (see Section 27.2).

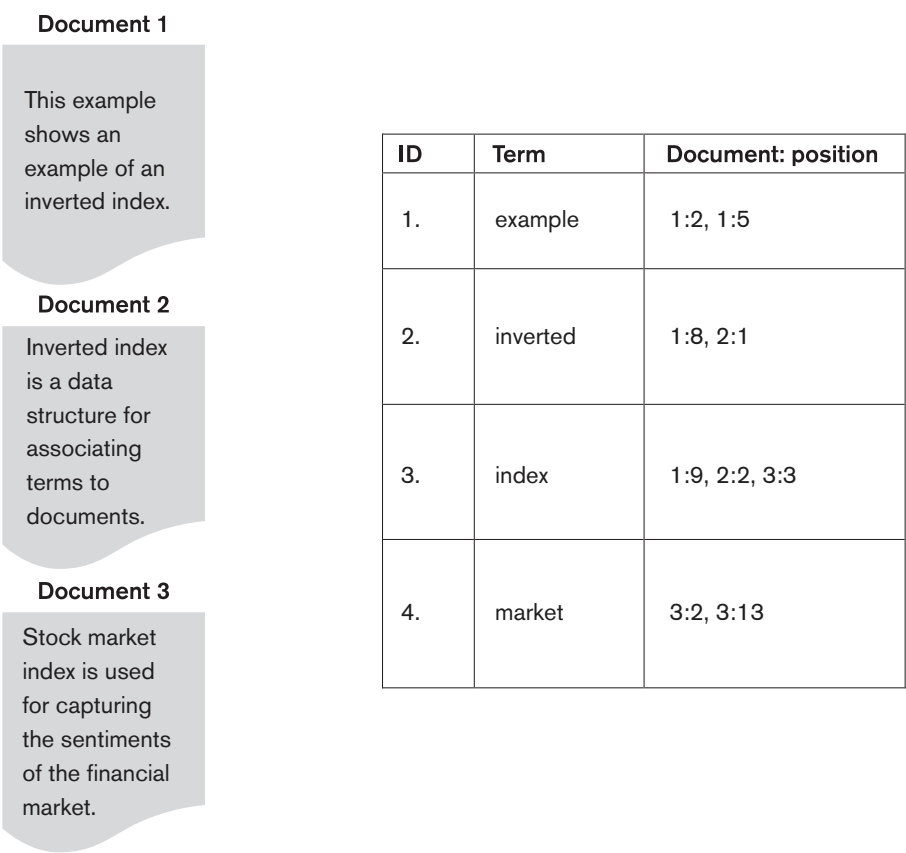
An **inverted index** of a document collection is a data structure that attaches distinct terms with a list of all documents that contains the term. The process of inverted index construction involves the extraction and processing steps shown in Figure 27.2. Acquired text is first preprocessed and the documents are represented with the vocabulary terms. Documents' statistics are collected in document lookup tables. Statistics generally include counts of vocabulary terms in individual documents as well as different collections, their positions of occurrence within the documents, and the lengths of the documents. The vocabulary terms are weighted at indexing time according to different criteria for collections. For example, in some cases terms in the titles of the documents may be weighted more heavily than terms that occur in other parts of the documents.

One of the most popular weighting schemes is the TF-IDF (term frequency-inverse document frequency) metric that we described in Section 27.2. For a given term, this weighting scheme distinguishes to some extent the documents in which the term occurs more often from those in which the term occurs very little or never. These weights are normalized to account for varying document lengths, further ensuring that longer documents with proportionately more occurrences of a word are not favored for retrieval over shorter documents with proportionately fewer occurrences. These processed document-term streams (matrices) are then inverted into term-document streams (matrices) for further IR steps.

Figure 27.4 shows an illustration of term-document-position vectors for the four illustrative terms—*example*, *inverted*, *index*, and *market*—which shows the positions where each term occurs in the three documents.

The steps involved in inverted index construction can be summarized as follows:

1. Break the documents into vocabulary terms by tokenizing, cleansing, removing stopwords, stemming, and/or using an additional thesaurus as vocabulary.
2. Collect document statistics and store the statistics in a document lookup table.
3. Invert the document-term stream into a term-document stream along with additional information such as term frequencies, term positions, and term weights.



**Figure 27.4**  
Example of an inverted index.

Searching for relevant documents from the inverted index, given a set of query terms, is generally a three-step process.

- 1. **Vocabulary search.** If the query comprises multiple terms, they are separated and treated as independent terms. Each term is searched in the vocabulary. Various data structures, like variations of B<sup>+</sup>-tree or hashing, may be used to optimize the search process. Query terms may also be ordered in lexicographic order to improve space efficiency.
- 2. **Document information retrieval.** The document information for each term is retrieved.
- 3. **Manipulation of retrieved information.** The document information vector for each term obtained in step 2 is now processed further to incorporate various forms of query logic. Various kinds of queries like prefix, range, context, and proximity queries are processed in this step to construct the final result based on the document collections returned in step 2.

### 27.5.1 Introduction to Lucene

Lucene is an actively maintained open source indexing/search engine that has become popular in both academic and commercial settings. Indexing is the primary focus of Lucene, but it uses indexing to facilitate search. The Lucene library is written in Java and comes with out-of-the-box scalable and high-performance capability. Lucene is the engine that powers another widely popular enterprise search application called Solr.<sup>25</sup> Solr provides many add-on capabilities to Lucene, such as providing Web interfaces for indexing many different document formats.

An upcoming book by Moczar (2015) discusses both Lucene and Solr.

**Indexing:** In Lucene, documents must go through a process of indexing before they become available for search. A Lucene document is made up of a set of fields. Fields hold the type of data in the index and are loosely comparable to columns in a database table. A field can be of type binary, numeric, or text data. Text fields consist of either entire chunk of untokenized text or a series of processed lexical units called token streams. The token streams are created via application of different types of available tokenization and filtering algorithms. For example, StandardTokenizer is one of the available tokenizers in Lucene that implements Unicode text segmentation for splitting words apart. There are other tokenizers, such as a WhitespaceTokenizer, that divide text at whitespaces. It is also easy to extend these tokenizers and filters in Lucene to create custom text analysis algorithms for tokenization and filtering. These analysis algorithms are central to achieving desired search results. Lucene provides APIs and several implementations for many high-speed and efficient tokenization and filtering algorithms. These algorithms have been extended for several different languages and domains, and they feature implementations of natural language processing algorithms for stemming, conducting dictionary-driven lemmatization, performing morphological analysis, conducting phonetic analysis, and so on.

**Search:** With a powerful search API, queries are matched against documents and a ranked list of results is retrieved. Queries are compared against the term vectors in inverted indexes to compute relevance scores based on the vector space model (see Section 27.2.2). Lucene provides a highly configurable search API wherein one can create queries for wildcard, exact, Boolean, proximity, and range searches. Lucene's default scoring algorithm uses variants of TF-IDF scoring to rank search results. To speed up search, Lucene maintains document-dependent normalization factors precomputed at index time; these are called norms of term vectors in document fields. These precomputed norms speed up the scoring process in Lucene. The actual query matching algorithms use functions that do very little computation at query matching time.

**Applications:** One of the reasons for Lucene's immense popularity is the ease of availability of Lucene applications for handling various document collections and

---

<sup>25</sup>See <http://lucene.apache.org/solr/>

deployment systems for indexing large unstructured document collections. The enterprise search application built on top of Lucene is called Solr. Solr is a Web server application that provides support for faceted search (see Section 27.8.1 on faceted search), custom format document processing support (such as PDF, HTML, etc.), and Web services for several API functions for indexing and search in Lucene.

## 27.6 Evaluation Measures of Search Relevance

Without proper evaluation techniques, one cannot compare and measure the relevance of different retrieval models and IR systems in order to make improvements. Evaluation techniques of IR systems measure the *topical relevance* and *user relevance*. **Topical relevance** measures the extent to which the topic of a result matches the topic of the query. Mapping one's information need with "perfect" queries is a cognitive task, and many users are not able to effectively form queries that would retrieve results more suited to their information need. Also, since a major chunk of user queries are informational in nature, there is no fixed set of right answers to show to the user. **User relevance** is a term used to describe the "goodness" of a retrieved result with regard to the user's information need. User relevance includes other implicit factors, such as user perception, context, timeliness, the user's environment, and current task needs. Evaluating user relevance may also involve subjective analysis and study of user retrieval tasks to capture some of the properties of implicit factors involved in accounting for users' bias for judging performance.

In Web information retrieval, no binary classification decision is made on whether a document is relevant or nonrelevant to a query (whereas the Boolean (or binary) retrieval model uses this scheme, as we discussed in Section 27.2.1). Instead, a ranking of the documents is produced for the user. Therefore, some evaluation measures focus on comparing different rankings produced by IR systems. We discuss some of these measures next.





### 27.6.1 Recall and Precision

Recall and precision metrics are based on the binary relevance assumption (whether each document is relevant or nonrelevant to the query). **Recall** is defined as the number of relevant documents retrieved by a search divided by the total number of actually relevant documents existing in the database. **Precision** is defined as the number of relevant documents retrieved by a search divided by the total number of documents retrieved by that search. Figure 27.5 is a pictorial representation of the terms *retrieved* versus *relevant* and shows how search results relate to four different sets of documents.

The notation for Figure 27.5 is as follows:

- TP: true positive
- FP: false positive



		Relevant?	
		Yes	No
Retrieved?	Yes	 Hits TP	 False Alarms FP
	No	Misses  FN	Correct Rejections TN 

**Figure 27.5**

Retrieved versus relevant search results.

- FN: false negative
- TN: true negative

The terms *true positive*, *false positive*, *false negative*, and *true negative* are generally used in any type of classification tasks to compare the given classification of an item with the desired correct classification. Using the term *hits* for the documents that truly or “correctly” match the user request, we can define *recall* and *precision* as follows:

$$\text{Recall} = |\text{Hits}|/|\text{Relevant}|$$

$$\text{Precision} = |\text{Hits}|/|\text{Retrieved}|$$

Recall and precision can also be defined in a ranked retrieval setting. Let us assume that there is one document at each rank position. The recall at rank position  $i$  for document  $d_i^q$  (denoted by  $r(i)$ ) ( $d_i^q$  is the retrieved document at position  $i$  for query  $q$ ) is the fraction of relevant documents from  $d_1^q$  to  $d_i^q$  in the result set for the query. Let the set of relevant documents from  $d_1^q$  to  $d_i^q$  in that set be  $S_i$  with cardinality  $|S_i|$ . Let  $(|D_q|)$  be the size of relevant documents for the query. In this case,  $|S_i| \leq |D_q|$ . Then:

$$\text{Ranked\_retrieval\_recall: } r(i) = |S_i|/|D_q|$$

The precision at rank position  $i$  or document  $d_i^q$  (denoted by  $p(i)$ ) is the fraction of documents from  $d_1^q$  to  $d_i^q$  in the result set that are relevant:

$$\text{Ranked\_retrieval\_precision: } p(i) = |S_i|/i$$

Table 27.2 illustrates the  $p(i)$ ,  $r(i)$ , and average precision (discussed in the next section) metrics. It can be seen that recall can be increased by presenting more results to the user, but this approach runs the risk of decreasing the precision. In the example, the number of relevant documents for some query = 10. The rank position and the relevance of an individual document are shown. The precision and recall value can be computed at each position within the ranked list as shown in the last two columns. As we see in Table 27.2, the ranked\_retrieval\_recall rises monotonically whereas the precision is prone to fluctuation.

**Table 27.2** Precision and Recall for Ranked Retrieval

Doc. No.	Rank Position $i$	Relevant	Precision( $i$ )	Recall( $i$ )
10	1	Yes	1/1 = 100%	1/10 = 10%
2	2	Yes	2/2 = 100%	2/10 = 20%
3	3	Yes	3/3 = 100%	3/10 = 30%
5	4	No	3/4 = 75%	3/10 = 30%
17	5	No	3/5 = 60%	3/10 = 30%
34	6	No	3/6 = 50%	3/10 = 30%
215	7	Yes	4/7 = 57.1%	4/10 = 40%
33	8	Yes	5/8 = 62.5%	5/10 = 50%
45	9	No	5/9 = 55.5%	5/10 = 50%
16	10	Yes	6/10 = 60%	6/10 = 60%

### 27.6.2 Average Precision

Average precision is computed based on the precision at each relevant document in the ranking. This measure is useful for computing a single precision value to compare different retrieval algorithms on a query  $q$ .

$$P_{\text{avg}} = \sum_{d_i^r \in D_q} p(i) / |D_q|$$

Consider the sample precision values of relevant documents in Table 27.2. The average precision ( $P_{\text{avg}}$  value) for the example in Table 27.2 is  $P(1) + P(2) + P(3) + P(7) + P(8) + P(10)/6 = 79.93\%$  (only relevant documents are considered in this calculation). Many good algorithms tend to have high top- $k$  average precision for small values of  $k$ , with correspondingly low values of recall.

### 27.6.3 Recall/Precision Curve

A recall/precision curve can be drawn based on the recall and precision values at each rank position, where the  $x$ -axis is the recall and the  $y$ -axis is the precision. Instead of using the precision and recall at each rank position, the curve is commonly plotted using recall levels  $r(i)$  at 0%, 10%, 20% ... 100%. The curve usually has a negative slope, reflecting the inverse relationship between precision and recall.

### 27.6.4 F-Score

$F$ -score ( $F$ ) is the harmonic mean of the precision ( $p$ ) and recall ( $r$ ) values. That is,

$$\frac{1}{F} = \frac{\frac{1}{p} + \frac{1}{r}}{2}$$

High precision is achieved almost always at the expense of recall and vice versa. It is a matter of the application's context whether to tune the system for high precision or high recall.  $F$ -score is typically used as a single measure that combines precision and recall to compare different result sets:

$$F = \frac{2pr}{p+r}$$

One of the properties of harmonic mean is that the harmonic mean of two numbers tends to be closer to the smaller of the two. Thus  $F$  is automatically biased toward the smaller of the precision and recall values. Therefore, for a high  $F$ -score, both precision and recall must be high.

$$F = \frac{2}{\frac{1}{p} + \frac{1}{r}}$$

## 27.7 Web Search and Analysis<sup>26</sup>

The emergence of the Web has brought millions of users to search for information, which is stored in a very large number of active sites. To make this information accessible, search engines such as Google, Bing and Yahoo! must crawl and index these sites and document collections in their index databases. Moreover, search engines must regularly update their indexes given the dynamic nature of the Web as new Web sites are created and current ones are updated or deleted. Since there are many millions of pages available on the Web on different topics, search engines must apply many sophisticated techniques such as link analysis to identify the importance of pages.

There are other types of search engines besides the ones that regularly crawl the Web and create automatic indexes: these are human-powered, vertical search engines or metasearch engines. These search engines are developed with the help of computer-assisted systems to aid the curators with the process of assigning indexes. They consist of manually created specialized Web directories that are hierarchically organized indexes to guide user navigation to different resources on the Web. **Vertical search engines** are customized topic-specific search engines that crawl and index a specific collection of documents on the Web and provide search results from that specific collection. **Metasearch engines** are built on top of search engines: they query different search engines simultaneously and aggregate and provide search results from these sources.

Another source of searchable Web documents is digital libraries. **Digital libraries** can be broadly defined as collections of electronic resources and services for the delivery of materials in a variety of formats. These collections may include a university's library catalog, catalogs from a group of participating universities, as in the

---

<sup>26</sup>The contribution of Pranesh P. Ranganathan and Hari P. Kumar to this section is appreciated.

State of Florida University System, or a compilation of multiple external resources on the World Wide Web, such as Google Scholar or the IEEE/ACM index. These interfaces provide universal access to different types of content—such as books, articles, audio, and video—situated in different database systems and remote repositories. Similar to real libraries, these digital collections are maintained via a catalog and organized in categories for online reference. Digital libraries “include personal, distributed, and centralized collections such as online public-access catalogs (OPACs) and bibliographic databases, distributed document databases, scholarly and professional discussion lists and electronic journals, other online databases, forums, and bulletin boards.”<sup>27</sup>

### 27.7.1 Web Analysis and Its Relationship to Information Retrieval

In addition to browsing and searching the Web, another important activity closely related to information retrieval is to *analyze* or *mine* information on the Web for new information of interest. (We discuss mining of data from files and databases in Chapter 28.) Application of data analysis techniques for discovery and analysis of useful information from the Web is known as **Web analysis**. Over the past few years, the World Wide Web has emerged as an important repository of information for many day-to-day applications for individual consumers, as well as a significant platform for e-commerce and for social networking. These properties make it an interesting target for data analysis applications. The Web mining and analysis field is an integration of a wide range of fields spanning information retrieval, text analysis, natural language processing, data mining, machine learning, and statistical analysis.

The goals of Web analysis are to improve and personalize search results relevance and to identify trends that may be of value to various businesses and organizations. We elaborate on these goals next.

- **Finding relevant information.** People usually search for specific information on the Web by entering keywords in a search engine or browsing information portals and using services. Search services are heavily constrained by search relevance problems since search engines must map and approximate the information need of millions of users as an *a priori* task. Low *precision* (see Section 27.6) ensues due to results that are nonrelevant to the user. In the case of the Web, high *recall* (see Section 27.6) is impossible to determine due to the inability to index all the pages on the Web. Also, measuring recall does not make sense since the user is concerned with only the top few documents. The most relevant results for the user are typically from only the top few results.
- **Personalization of the information.** Different people have different content and presentation preferences. Various customization tools used in

---

<sup>27</sup>Covi and Kling (1996), page 672.

Web-based applications and services (such as click-through monitoring, eyeball tracking, explicit or implicit user profile learning, and dynamic service composition using Web APIs) are used for service adaptation and personalization. A personalization engine typically has algorithms that make use of the user's personalization information—collected by various tools—to generate user-specific search results. The Web has become a rich landscape where people leave traces as they navigate, click, like, comment, and buy things in this virtual space. This information is of high commercial value, and many companies in all kinds of consumer goods mine and sell this information for customer targeting.

- **Finding information of social value.** With more than 1 billion downloads of the Facebook app on various Android devices, one can imagine how popular the various social networks have become in recent times. People build what is called social capital in these virtual worlds such as Twitter and Facebook. **Social capital** refers to features of social organizations, such as networks, norms, and social trust, that facilitate coordination and cooperation for mutual benefit. Social scientists are studying social capital and how to harness this rich resource to benefit society in various ways. We briefly touch upon aspects of social search in Section 27.8.2.

Web analysis can be further classified into three categories: **Web structure analysis**, which discovers knowledge from hyperlinks that represent the structure of the Web; **Web content analysis**, which deals with extracting useful information/knowledge from Web page contents; and **Web usage analysis**, which mines user access patterns from usage logs that record the activity of every user.

### 27.7.2 Web Structure Analysis

The World Wide Web is a huge corpus of information, but locating resources that are both high quality and relevant to the needs of the user is very difficult. The set of Web pages taken as a whole has almost no unifying structure, with variability in authoring style and content; this variability makes it difficult to precisely locate needed information. Index-based search engines have been one of the primary tools by which users search for information on the Web. Web search engines **crawl** the Web and create an index to the Web for searching purposes. When a user specifies her need for information by supplying keywords, these Web search engines query their repository of indexes and produce links or URLs with abbreviated content as search results. There may be thousands of pages relevant to a particular query. A problem arises when only a few most relevant results are returned to the user. Our discussions of querying and relevance-based ranking in IR systems in (see Sections 27.2 and 27.3) is applicable to Web search engines. These ranking algorithms explore the link structure of the Web.

Web pages, unlike standard text collections, contain connections to other Web pages or documents (via the use of hyperlinks), allowing users to browse from page to page. A **hyperlink** has two components: a **destination page** and an **anchor text** that describes the link. For example, a person can link to the Yahoo Web site on her

Web page with anchor text such as “My favorite Web site.” Anchor texts can be thought of as being implicit endorsements. They provide important latent human annotation. A person linking to other Web pages from her Web page is assumed to have some relation to those Web pages. Web search engines aim to distill results per their relevance and authority. There are many redundant hyperlinks, like the links to the homepage on every Web page of the Web site. Such hyperlinks must be eliminated from the search results by the search engines.

A **hub** is a Web page or a Web site that links to a collection of prominent sites (authorities) on a common topic. A good **authority** is a page that is pointed to by many good hubs, whereas a good hub is a page that points to many good authorities. These ideas are used by the HITS ranking algorithm. We briefly discuss a couple of ranking algorithms in the next section.

### 27.7.3 Analyzing the Link Structure of Web Pages

The goal of **Web structure analysis** is to generate a structural representation about the Web site and Web pages. Web structure analysis focuses on the inner structure of documents and deals with the link structure using hyperlinks at the interdocument level. The structure and content of Web pages are often combined for information retrieval by Web search engines. Given a collection of interconnected Web documents, interesting and informative facts describing their connectivity in the Web subset can be discovered. Web structure analysis is also used to help with navigation and make it possible to compare/integrate different Web page schemes. This aspect of Web structure analysis facilitates Web document classification and clustering on the basis of structure.

**The PageRank Ranking Algorithm.** As discussed earlier, ranking algorithms are used to order search results based on relevance and authority. Google uses the well-known **PageRank** algorithm,<sup>28</sup> which is based on the “importance” of each page. Every Web page has a number of forward links (out-edges) and backlinks (in-edges). It is very difficult to determine all the backlinks of a Web page, whereas it is relatively straightforward to determine its forward links. According to the PageRank algorithm, highly linked pages are more important (have greater authority) than pages with fewer links. However, not all backlinks are important. A backlink to a page from a credible source is more important than a link from some arbitrary page. Thus a page has a high rank if the sum of the ranks of its backlinks is high. PageRank was an attempt to see how good an approximation of the “importance” of a page can be obtained from the link structure.

The computation of page ranking follows an iterative approach. PageRank of a Web page is calculated as a sum of the PageRanks of all its backlinks. PageRank treats the Web like a *Markov model*. An imaginary Web surfer visits an infinite string of pages by clicking randomly. The PageRank of a page is an estimate of how often the surfer winds

---

<sup>28</sup>The PageRank algorithm was proposed by Lawrence Page (1998) and Sergey Brin, founders of Google. For more information, see <http://en.wikipedia.org/wiki/PageRank>

up at a particular page. PageRank is a measure of the query-independent importance of a page/node. For example, let  $P(X)$  be the PageRank of any page  $X$  and  $C(X)$  be the number of outgoing links from page  $X$ , and let  $d$  be the damping factor in the range  $0 < d < 1$ . Usually  $d$  is set to 0.85. Then PageRank for a page  $A$  can be calculated as:

$$P(A) = (1 - d) + d(P(T_1)/C(T_1) + P(T_2)/C(T_2) + \dots + P(T_n)/C(T_n))$$

Here  $T_1, T_2, \dots, T_n$  are the pages that point to Page  $A$  (that is, are citations to page  $A$ ). PageRank forms a probability distribution over Web pages, so the sum of all Web pages' PageRanks is one.

**The HITS Ranking Algorithm.** The HITS<sup>29</sup> algorithm proposed by Jon Kleinberg is another type of ranking algorithm exploiting the link structure of the Web. The algorithm presumes that a good hub is a document that points to many hubs, and a good authority is a document that is pointed at by many other authorities. The algorithm contains two main steps: a sampling component and a weight-propagation component. The sampling component constructs a focused collection  $S$  of pages with the following properties:

1.  $S$  is relatively small.
2.  $S$  is rich in relevant pages.
3.  $S$  contains most (or a majority) of the strongest authorities.

The weight component recursively calculates the hub and authority values for each document as follows:

1. Initialize hub and authority values for all pages in  $S$  by setting them to 1.
2. While (hub and authority values do not converge):
  - a. For each page in  $S$ , calculate authority value = Sum of hub values of all pages *pointing to* the current page.
  - b. For each page in  $S$ , calculate hub value = Sum of authority values of all pages *pointed at by* the current page.
  - c. Normalize hub and authority values such that sum of all hub values in  $S$  equals 1 and the sum of all authority values in  $S$  equals 1.

## 27.7.4 Web Content Analysis

As mentioned earlier, **Web content analysis** refers to the process of discovering useful information from Web content/data/documents. The **Web content data** consists of unstructured data such as free text from electronically stored documents, semi-structured data typically found as HTML documents with embedded image data, and more structured data such as tabular data and pages in HTML, XML, or other markup languages generated as output from databases. More generally, the term *Web content* refers to any real data in the Web page that is intended for the user accessing that page. This usually consists of but is not limited to text and graphics.

---

<sup>29</sup>See Kleinberg (1999).



We will first discuss some preliminary Web content analysis tasks and then look at the traditional analysis tasks of Web page classification and clustering.

**Structured Data Extraction.** Structured data on the Web is often very important because it represents essential information, such as a structured table showing the airline flight schedule between two cities. There are several approaches to structured data extraction. One includes writing a **wrapper**, or a program that looks for different structural characteristics of the information on the page and extracts the right content. Another approach is to manually write an extraction program for each Web site based on observed format patterns of the site, which is very labor intensive and time consuming. This latter approach does not scale to a large number of sites. A third approach is **wrapper induction** or **wrapper learning**, where the user first manually labels a set of training set pages and the learning system generates rules—based on the learning pages—that are applied to extract target items from other Web pages. A fourth approach is the automatic approach, which aims to find patterns/grammars from the Web pages and then uses **wrapper generation** to produce a wrapper to extract data automatically.

**Web Information Integration.** The Web is immense and has billions of documents, authored by many different persons and organizations. Because of this, Web pages that contain similar information may have different syntax and different words that describe the same concepts. This creates the need for integrating information from diverse Web pages. Two popular approaches for Web information integration are:

1. **Web query interface integration**, to enable querying multiple Web databases that are not visible in external interfaces and are hidden in the “deep Web.” The **deep Web**<sup>30</sup> consists of those pages that do not exist until they are created dynamically as the result of a specific database search, which produces some of the information in the page (see Chapter 11). Since traditional search engine crawlers cannot probe and collect information from such pages, the deep Web has heretofore been hidden from crawlers.
2. **Schema matching**, such as integrating directories and catalogs to come up with a global schema for applications. An example of such an application would be to match and combine into one record data from various sources by cross-linking health records from multiple systems. The result would be an individual global health record.

These approaches remain an area of active research, and a detailed discussion of them is beyond the scope of this text. Consult the Selected Bibliography at the end of this chapter for further details.

**Ontology-Based Information Integration.** This task involves using ontologies to effectively combine information from multiple heterogeneous sources. Ontologies—formal models of representation with explicitly defined concepts and named

---

<sup>30</sup>The deep Web as defined by Bergman (2001).

relationships linking them—are used to address the issues of semantic heterogeneity in data sources. Different classes of approaches are used for information integration using ontologies.

- **Single ontology approaches** use one global ontology that provides a shared vocabulary for the specification of the semantics. They work if all information sources to be integrated provide nearly the same view on a domain of knowledge. For example, UMLS (described in Section 27.4.3) can serve as a common ontology for biomedical applications.
- In a **multiple ontology approach**, each information source is described by its own ontology. In principle, the “source ontology” can be a combination of several other ontologies, but it cannot be assumed that the different “source ontologies” share the same vocabulary. Dealing with multiple, partially overlapping, and potentially conflicting ontologies is a difficult problem faced by many applications, including those in bioinformatics and other complex topics of study.

**Building Concept Hierarchies.** One common way of organizing search results is via a linear ranked list of documents. But for some users and applications, a better way to display results would be to create groupings of related documents in the search result. One way of organizing documents in a search result, and for organizing information in general, is by creating a **concept hierarchy**. The documents in a search result are organized into groups in a hierarchical fashion. Other related techniques to organize documents are through **classification** and **clustering** (see Chapter 28). Clustering creates groups of documents, where the documents in each group share many common concepts.

**Segmenting Web Pages and Detecting Noise.** There are many superfluous parts in a Web document, such as advertisements and navigation panels. The information and text in these superfluous parts should be eliminated as noise before classifying the documents based on their content. Hence, before applying classification or clustering algorithms to a set of documents, the areas or blocks of the documents that contain noise should be removed.

### 27.7.5 Approaches to Web Content Analysis

The two main approaches to Web content analysis are (1) agent based (IR view) and (2) database based (DB view).

The **agent-based approach** involves the development of sophisticated artificial intelligence systems that can act autonomously or semi-autonomously on behalf of a particular user, to discover and process Web-based information. Generally, the agent-based Web analysis systems can be placed into the following three categories:

- **Intelligent Web agents** are software agents that search for relevant information using characteristics of a particular application domain (and possibly a user profile) to organize and interpret the discovered information. For

example, an intelligent agent retrieves product information from a variety of vendor sites using only general information about the product domain.

- **Information filtering/categorization** is another technique that utilizes Web agents for categorizing Web documents. These Web agents use methods from information retrieval, as well as semantic information based on the links among various documents, to organize documents into a concept hierarchy.
- **Personalized Web agents** are another type of Web agents that utilize the personal preferences of users to organize search results, or to discover information and documents that could be of value for a particular user. User preferences could be learned from previous user choices, or from other individuals who are considered to have similar preferences to the user.

The **database-based approach** aims to infer the structure of the Web site or to transform a Web site to organize it as a database so that better information management and querying on the Web become possible. This approach of Web content analysis primarily tries to model the data on the Web and integrate it so that more sophisticated queries than keyword-based search can be performed. These could be achieved by finding the schema of Web documents or building a Web document warehouse, a Web knowledge base, or a virtual database. The database-based approach may use a model such as the Object Exchange Model (OEM),<sup>31</sup> which represents semistructured data by a labeled graph. The data in the OEM is viewed as a graph, with objects as the vertices and labels on the edges. Each object is identified by an object identifier and a value that is either atomic—such as integer, string, GIF image, or HTML document—or complex in the form of a set of object references.

The main focus of the database-based approach has been with the use of multilevel databases and Web query systems. A **multilevel database** at its lowest level is a database containing primitive semistructured information stored in various Web repositories, such as hypertext documents. At the higher levels, metadata or generalizations are extracted from lower levels and organized in structured collections such as relational or object-oriented databases. In a **Web query system**, information about the content and structure of Web documents is extracted and organized using database-like techniques. Query languages similar to SQL can then be used to search and query Web documents. These types of queries combine structural queries, based on the organization of hypertext documents, and content-based queries.

### 27.7.6 Web Usage Analysis

**Web usage analysis** is the application of data analysis techniques to discover usage patterns from Web data, in order to understand and better serve the needs of Web-based applications. This activity does not directly contribute to information retrieval; but it is important for improving and enhancing users' search experiences.

---

<sup>31</sup>See Kosala and Blockeel (2000).

**Web usage data** describes the pattern of usage of Web pages, such as IP addresses, page references, and the date and time of accesses for a user, user group, or an application. Web usage analysis typically consists of three main phases: preprocessing, pattern discovery, and pattern analysis.

1. **Preprocessing.** Preprocessing converts the information collected about usage statistics and patterns into a form that can be utilized by the pattern discovery methods. For example, we use the term *page view* to refer to pages viewed or visited by a user. There are several different types of preprocessing techniques available:
  - **Usage preprocessing** analyzes the available collected data about usage patterns of users, applications, and groups of users. Because this data is often incomplete, the process is difficult. Data cleaning techniques are necessary to eliminate the impact of irrelevant items in the analysis result. Frequently, usage data is identified by an IP address and consists of clicking streams that are collected at the server. Better data is available if a usage tracking process is installed at the client site.
  - **Content preprocessing** is the process of converting text, image, scripts, and other content into a form that can be used by the usage analysis. Often, this process consists of performing content analysis such as classification or clustering. The clustering or classification techniques can group usage information for similar types of Web pages, so that usage patterns can be discovered for specific classes of Web pages that describe particular topics. Page views can also be classified according to their intended use, such as for sales or for discovery or for other uses.
  - **Structure preprocessing** can be done by parsing and reformatting the information about hyperlinks and structure between viewed pages. One difficulty is that the site structure may be dynamic and may have to be constructed for each server session.
2. **Pattern discovery.** The techniques that are used in pattern discovery are based on methods from the fields of statistics, machine learning, pattern recognition, data analysis, data mining, and other similar areas. These techniques are adapted so they take into consideration the specific knowledge and characteristics of Web analysis. For example, in association rule discovery (see Section 28.2), the notion of a transaction for market-basket analysis considers the items to be unordered. But the order of accessing of Web pages is important, and so it should be considered in Web usage analysis. Hence, pattern discovery involves mining sequences of page views. In general, using Web usage data, the following types of data mining activities may be performed for pattern discovery.
  - **Statistical analysis.** Statistical techniques are the most common method of extracting knowledge about visitors to a Web site. By analyzing the session log, it is possible to apply statistical measures such as mean, median, and frequency count to parameters such as pages viewed, viewing time per page, length of navigation paths between pages, and other parameters that are relevant to Web usage analysis.

- **Association rules.** In the context of Web usage analysis, association rules refer to sets of pages that are accessed together with a support value exceeding some specified threshold. (See Section 28.2 on association rules.) These pages may not be directly connected to one another via hyperlinks. For example, association rule discovery may reveal a correlation between users who visited a page containing electronic products to those who visit a page about sporting equipment.
  - **Clustering.** In the Web usage domain, there are two kinds of interesting clusters to be discovered: usage clusters and page clusters. **Clustering of users** tends to establish groups of users exhibiting similar browsing patterns. Such knowledge is especially useful for inferring user demographics in order to perform market segmentation in e-commerce applications or provide personalized Web content to the users. **Clustering of pages** is based on the content of the pages, and pages with similar contents are grouped together. This type of clustering can be utilized in Internet search engines and in tools that provide assistance to Web browsing.
  - **Classification.** In the Web domain, one goal is to develop a profile of users belonging to a particular class or category. This requires extraction and selection of features that best describe the properties of a given class or category of users. For example, an interesting pattern that may be discovered would be: 60% of users who placed an online order in /Product/Books are in the 18–25 age group and live in rented apartments.
  - **Sequential patterns.** These kinds of patterns identify sequences of Web accesses, which may be used to predict the next set of Web pages to be accessed by a certain class of users. These patterns can be used by marketers to produce targeted advertisements on Web pages. Another type of sequential pattern pertains to which items are typically purchased following the purchase of a particular item. For example, after purchasing a computer, a printer is often purchased.
  - **Dependency modeling.** Dependency modeling aims to determine and model significant dependencies among the various variables in the Web domain. For example, one may be interested in building a model that represents the various stages a visitor undergoes while shopping in an online store; this model would be based on user actions (e.g., being a casual visitor versus being a serious potential buyer).
3. **Pattern analysis.** The final step is to filter out those rules or patterns that are considered to be not of interest based on the discovered patterns. One common technique for pattern analysis is to use a query language such as SQL to detect various patterns and relationships. Another technique involves loading usage data into a data warehouse with ETL tools and performing OLAP operations to view the data along multiple dimensions (see Section 29.3). It is common to use visualization techniques, such as graphing patterns or assigning colors to different values, to highlight patterns or trends in the data.

### 27.7.7 Practical Applications of Web Analysis

**Web Analytics.** The goal of **web analytics** is to understand and optimize the performance of Web usage. This requires collecting, analyzing, and monitoring the performance of Internet usage data. On-site Web analytics measures the performance of a Web site in a commercial context. This data is typically compared against key performance indicators to measure effectiveness or performance of the Web site as a whole, and it can be used to improve a Web site or improve the marketing strategies.

**Web Spamming.** It has become increasingly important for companies and individuals to have their Web sites/Web pages appear in the top search results. To achieve this, it is essential to understand search engine ranking algorithms and to present the information in one's page in such a way that the page is ranked high when the respective keywords are queried. There is a thin line separating legitimate page optimization for business purposes and spamming. **Web spamming** is thus defined as a deliberate activity to promote one's page by manipulating the results returned by the search engines. Web analysis may be used to detect such pages and discard them from search results.

**Web Security.** Web analysis can be used to find interesting usage patterns of Web sites. If any flaw in a Web site has been exploited, it can be inferred using Web analysis, thereby allowing the design of more robust Web sites. For example, the backdoor or information leak of Web servers can be detected by using Web analysis techniques on abnormal Web application log data. Security analysis techniques such as intrusion detection and denial-of-service attacks are based on Web access pattern analysis.

**Web Crawlers.** These are programs that visit Web pages and create copies of all the visited pages so they can be processed by a search engine for indexing the downloaded pages and providing fast searches. Another use of crawlers is to automatically check and maintain Web sites. For example, the HTML code and the links in a Web site can be checked and validated by the crawler. Another unfortunate use of crawlers is to collect e-mail addresses and other personal information from Web pages; the information is subsequently used in sending spam e-mails.

## 27.8 Trends in Information Retrieval

In this section, we review a few concepts that are being considered in recent research work in information retrieval.

### 27.8.1 Faceted Search

Faceted search is a technique that allows for an integrated search and navigation experience by allowing users to explore by filtering available information. This search technique is often used in ecommerce Web sites and applications and

enables users to navigate a multi-dimensional information space. Facets are generally used for handling three or more dimensions of classification. These multiple dimensions of classification allow the **faceted classification scheme** to classify an object in various ways based on different taxonomical criteria. For example, a Web page may be classified in various ways: by content (airlines, music, news, etc.); by use (sales, information, registration, etc.); by location; by language used (HTML, XML, etc.); and in other ways or facets. Hence, the object can be classified in multiple ways based on multiple taxonomies.

A **facet** defines properties or characteristics of a class of objects. The properties should be mutually exclusive and exhaustive. For example, a collection of art objects might be classified using an artist facet (name of artist), an era facet (when the art was created), a type facet (painting, sculpture, mural, etc.), a country of origin facet, a media facet (oil, watercolor, stone, metal, mixed media, etc.), a collection facet (where the art resides), and so on.

Faceted search uses faceted classification, which enables a user to navigate information along multiple paths corresponding to different orderings of the facets. This contrasts with traditional taxonomies, in which the hierarchy of categories is fixed and unchanging. University of California–Berkeley’s Flamenco project<sup>32</sup> is one of the earlier examples of a faceted search system. Most e-commerce sites today, such as Amazon or Expedia, use faceted search in their search interfaces to quickly compare and navigate various aspects related to search criteria.

## 27.8.2 Social Search

The traditional view of Web navigation and browsing assumes that a single user is searching for information. This view contrasts with previous research by library scientists who studied users’ information-seeking habits. This research demonstrated that additional individuals may be valuable information resources during information search by a single user. More recently, research indicates that there is often direct user cooperation during Web-based information search. Some studies report that significant segments of the user population are engaged in explicit collaboration on joint search tasks on the Web. Active collaboration by multiple parties also occurs in certain cases (for example, enterprise settings); at other times, and perhaps for a majority of searches, users often interact with others remotely, asynchronously, and even involuntarily and implicitly.

Socially enabled online information search (social search) is a new phenomenon facilitated by recent Web technologies. **Collaborative social search** involves different ways for active involvement in search-related activities such as co-located search, remote collaboration on search tasks, use of social network for search, use of expertise networks, use of social data mining or collective intelligence to improve the search process, and use of social interactions to facilitate information seeking and sense making. This social search activity may be done synchronously, asynchronously,

---

<sup>32</sup>Yee (2003) describes faceted metadata for image search.



co-located, or in remote shared workspaces. Social psychologists have experimentally validated that the act of social discussions has facilitated cognitive performance. People in social groups can provide solutions (answers to questions), pointers to databases or to other people (meta-knowledge), and validation and legitimization of ideas; in addition, social groups can serve as memory aids and can help with problem reformulation. **Guided participation** is a process in which people co-construct knowledge in concert with peers in their community. Information seeking is mostly a solitary activity on the Web today. Some recent work on collaborative search reports several interesting findings and the potential of this technology for better information access. It is increasingly common for people to use social networks such as Facebook to seek opinions and clarifications on various topics and to read product reviews before making a purchase.

### 27.8.3 Conversational Information Access

**Conversational information access** is an interactive and collaborative information-finding interaction. The participants engage in a natural human-to-human conversation, and intelligent agents listen to the conversation in the background and perform **intent extraction** to provide participants with need-specific information. Agents use direct or subtle interactions with participants via mobile or wearable communication devices. These interactions require technologies like speaker identification, keyword spotting, automatic speech recognition, semantic understanding of conversations, and discourse analysis as a means of providing users with faster and relevant pointers for conversations. Via technologies like those just mentioned, information access is transformed from a solitary activity to a participatory activity. In addition, information access becomes more goal specific as agents use multiple technologies to gather relevant information and as participants provide conversational feedback to agents.

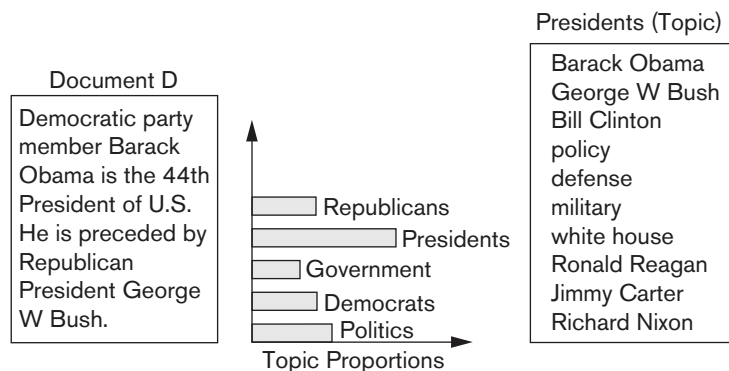
### 27.8.4 Probabilistic Topic Modeling

The unprecedented growth in information generated with the advent of the Web has led to issues concerning how to organize data into categories that will facilitate correct and efficient dissemination of information. For example, international news agencies like Reuters and the Associated Press gather daily news worldwide pertaining to business, sports, politics, technology, and so on. It is a tremendous challenge to organize effectively this plethora of information. Search engines have conventionally organized words within and links among documents to make them accessible on the Web. Organizing information according to the topics and themes of documents allows users to navigate through the vast amount of information based on the topics they are interested in.

To address this problem, a class of machine learning algorithms known as **probabilistic topic models** has emerged in the last decade. These algorithms can automatically organize large collections of documents into relevant themes. The beauty of these algorithms is that they are totally unsupervised, meaning that they

**Figure 27.6**

A document D and its topic proportions.



do not need any training sets or human annotations to perform this thematic extrapolation. The concept of this class of algorithms is as follows: Every document is inherently organized thematically. For example, documents about Barack Obama may mention other presidents, other issues related to the government, or a particular political theme. An article about one of the *Iron Man* movies may contain references to other sci-fi (science fiction) characters from the Marvel series or generally have a sci-fi theme. These inherent structures in documents can be extracted by probabilistic modeling and estimation methods. As another example, let us assume that every document is made up of a collection of different topics in differing proportions (e.g., a document about politics may also be about presidents and American history). Also, every topic is made up of a collection of words.

By considering Figure 27.6, we can guess that document D, which mentions U.S. Presidents Barack Obama and George W. Bush, can belong to the topics Presidents, Politics, Democrats, Republicans, and Government. In general, topics share a fixed vocabulary of words. This vocabulary of words is extracted from the collection of documents for which we wish to train the topic models. We generally choose the number of topics we wish to extract from the collection. Every topic ranks words differently according to how often a word is represented under a certain topic in different documents. In Figure 27.6, the bars representing topic proportions should all sum to 1. Document D primarily belongs to the topic Presidents, as shown in the bar graph. Figure 27.6 depicts the topics related to Presidents along with the list of words associated with this topic.

Probabilistic topic modeling estimates topic distributions using a learning algorithm that assumes that documents can be generated as a mixture of topic proportions. These topic proportion estimates are computed using sampling and expectation maximization algorithms. An algorithm called latent Dirichlet allocation (LDA)<sup>33</sup> is used to generate the topic models. The model assumes a generative process wherein documents are mixtures of latent topics and topics are distributions over words. A generative model randomly generates observable data given

<sup>33</sup>See Blei, Ng, and Jordan (2003).

some hidden parameters. These hidden/unobserved parameters are the Dirichlet distribution<sup>34</sup> priors for words and topics, topic distributions, and per-topic word distributions. Bayesian inference methods such as Gibbs sampling<sup>35</sup> are used to fit the hidden parameters based on the observed data (the words in the documents).

### 27.8.5 Question Answering Systems

Question answering (QA) has become a hot topic of study due to the surge in virtual assistant technology (e.g., Apple's Siri and Microsoft's Cortana). These virtual assistant technologies are advancements in interactive voice response (IVR) systems, which primarily rely on speech recognition techniques such as keyword spotting. Question answering deals with complex understanding of natural language queries. Recently, IBM created history by developing the QA system called Watson, that participated in the *Jeopardy!* Challenge<sup>36</sup> and defeated human players in the popular TV quiz show. Question answering has emerged as a practical engineering discipline that comprises techniques such as parsing; named entity recognition (NER); focus extraction; answer type extraction; relation extraction; ontological inference; and search, indexing, and classification algorithms. Question answering techniques also involve knowledge engineering from large unstructured corpora such as Web document collections and structured databases that incorporate knowledge from various domains. These document collections are generally large enough to require application of big data tools and technologies, some of which we discussed in Chapter 25. In the following sections, we consider the main concepts involved in question answering.

**Types of Questions:** In question answering systems, it is important to know the category or type of question, because answering strategies rely heavily on the type of questions. Some of these categories are not always mutually exclusive and hence require hybrid answering strategies. Generally, questions can be categorized into the following types:

**Factoid Questions:** This type of question pinpoints the right phrase in a document or a database that correctly addresses the question. Examples of this type include questions such as, “Who is the president of the United States?”, “In which city was Elvis Presley born?”, “Where is Hartsfield Jackson International Airport located?”, and “At what time will today's sunset occur?”.

**List Questions:** This type of question seeks a list of factoid responses that satisfy a given criterion. Examples include “Name three plays that were written by Shakespeare”, “Name the male actors who played the role of James Bond in the James Bond 007 movie series”, and “List three red-colored vegetables”.

---

<sup>34</sup>S. Kotz, N. Balakrishnan, and N. L. Johnson (2000).

<sup>35</sup>German and German (1984).

<sup>36</sup>See Ferrucci et al. (2010).

**Definition Questions:** This type of question asks about the definition and meaning of the concept, and to extract the essential information and properties of the concept. Examples include “What is an inert gas?”, “Who is Alexander the Great?”, and “What is the LIBOR rate?”.

**Opinion Questions:** This type of question seeks different views on a subject that the question. For example, “What countries should be allowed to test nuclear weapons?” and “What is the sentiment in Saudi Arabia about terrorism in the Middle East?”

In recent years, joint initiatives in research and academia have advocated adopting common metrics, architectures, tools, and methodologies to create baselines that will facilitate and improve the QA technique.

**Architectures.** Most state-of-the-art QA architectures are generally made up of pipelines that comprise the following stages:

**Question Analysis:** This stage involves analyzing questions and converting them to structural representations of analyzed text for processing by downstream components. Answer types are extracted from parsed representations of questions using some or all of the following techniques: shallow semantic parsing, focus detection, answer type classification, named entity recognition, and co-reference resolution.

- **Shallow semantic parsing:** The process of assigning surface-level markups to sentence structures via supervised machine learning methods. In general, frames are automatically instantiated for sentences by trying to match “WHO did WHAT to WHOM, WHEN, WHERE, WHY, and HOW” elements.
- **Focus detection:** In an image, certain things stand out whereas others remain in the background. We say that things that stand out are in focus. Similarly, in QA, questions have focus words that contain references to answers. For example, in the question “Which book of Shakespeare is a tragedy about lovers?”, the focus words “book of Shakespeare” can be instantiated with the rule “which X”, where X is a noun phrase in a sentence. QA systems use focus words to trigger directed searches and to aid in answer resolution.
- **Answer type classification:** This phase helps determine the categories of answers in QA. In the preceding example, the headword of the focus words, “book”, is the answer type for this question. Several machine learning techniques are applied in QA to determine the answer type of a question.
- **Named entity recognition:** Named entity recognition seeks to classify elements in text into predefined categories, such as person, place, animal, country, river, continent.
- **Co-reference resolution:** The task of co-reference resolution is about identifying multiple expressions in text that refer to the same thing. For example, in the sentence “John said that he wanted to go to the theater on Sunday”, the pronoun “he” refers to “John” and is a co-reference in text.

**Query Generation:** In this stage, the analyzed text is used to generate multiple queries using query normalization and expansion techniques for one or more underlying search engines in which the answers may be embedded. For example, in the question, “Which book of Shakespeare is about tragedy of lovers?”, the expanded queries can be “Shakespeare love story”, “novels of Shakespeare”, “tragic love story author Shakespeare”, “love story genre tragedy author Shakespeare”, and so on. Extracted keywords, answer types, synonyms information, and named entities are generally used in different combinations to create different queries.

**Search:** In this stage, the queries are sent to different search engines and relevant passages are retrieved. Search engines where searches are performed can be online, such as Google or Bing, and offline, such as Lucene or Indri.<sup>37</sup>

**Candidate Answer Generation:** Named entity extractors are used on retrieved passages and matched against desired answer types to come up with candidate answers. Depending on the desired granularity of the answer, candidate generation and answer type matching algorithms are applied (e.g., surface pattern matching and structural matching). In surface pattern matching, regular expression templates are instantiated with arguments from the question and matched against lexical chunks of retrieved passages to extract answers. For example, focus words are aligned with passages containing potential answers to extract answer candidates. In the sentence, “Romeo and Juliet is a tragic love story by Shakespeare”, the phrase “Romeo and Juliet” can simply replace “Which book” in the question, “Which book is a tragic love story by Shakespeare?”. In structural matching, questions and retrieved passages are parsed and aligned together using syntactic and semantic alignment to find answer candidates. A sentence such as, “Shakespeare wrote the tragic love story Romeo and Juliet” cannot be surface matched with the aforementioned question, but with correct parsing and alignment will structurally match with the question.

**Answer Scoring:** In this stage, confidence scores for the candidate answers are estimated. Similar answers are merged; knowledge sources can be reused to gather supporting evidence for different candidate answers.

## 27.9 Summary

In this chapter, we covered an important area called information retrieval (IR) that is closely related to databases. With the advent of the Web, unstructured data with text, images, audio, and video is proliferating at phenomenal rates. Although database management systems have a very good handle on structured data, the unstructured data containing a variety of data types is being stored mainly on ad hoc information repositories on the Web that are available for consumption primarily via IR systems. Google, Yahoo, and similar search engines are IR systems that make the advances in this field readily available for the average end user and give end users a richer and continually improving search experience.

<sup>37</sup><http://www.lemurproject.org/indri/>

We started in Section 27.1 by first introducing the field of IR in section 27.1.1 and comparing IR and database technologies in Section 27.1.2. A brief history of IR was presented in Section 27.1.3, and the query and browsing modes of interaction in IR systems were introduced in Section 27.1.4.

We presented in Section 27.2 the various retrieval models used in IR, including Boolean, vector space, probabilistic, and semantic models. These models allow us to measure whether a document is relevant to a user query and provide similarity measurement heuristics. In Section 27.3 we presented different types of queries—in addition to keyword-based queries, which dominate, there are other types, including Boolean, phrase, proximity, natural language, and others for which explicit support needs to be provided by the retrieval model. Text preprocessing is important in IR systems, and we discussed in Section 27.4 various activities like stopword removal, stemming, and the use of thesauruses. We then discussed the construction and use of inverted indexes in Section 27.5, which are at the core of IR systems and contribute to factors involving search efficiency. We then discussed in Section 27.6 various evaluation metrics, such as recall precision and *F*-score, to measure the goodness of the results of IR queries. The Lucene open source indexing and search engine and its extension called Solr was discussed. Relevance feedback was briefly addressed—it is important to modify and improve the retrieval of pertinent information for the user through his interaction and engagement in the search process.

We provided in Section 27.7 a somewhat detailed introduction to analysis of the Web as it relates to information retrieval. We divided this treatment into the analysis of content, structure, and usage of the Web. Web search was discussed, including an analysis of the Web link structure (Section 27.7.3), including an introduction to algorithms for ranking the results from a Web search such as PageRank and HITS. Finally, we briefly discussed current trends, including faceted search, social search, and conversational search. We also presented probabilistic modeling of topics of documents and a popular technique called latent Dirichlet allocation. We ended the chapter with a discussion of question answering systems (Section 27.7.5), which are becoming very popular and use tools like Siri from Apple and Cortana from Microsoft.

This chapter provided an introductory treatment of a vast field. The interested reader should refer to the end-of-chapter bibliography for specialized texts on information retrieval and search engines.

## Review Questions

- 27.1.** What is structured data and what is unstructured data? Give an example of each from your experience.
- 27.2.** Give a general definition of *information retrieval* (IR). What does information retrieval involve when we consider information on the Web?
- 27.3.** Discuss the types of data and the types of users in today's information retrieval systems.

- 27.4. What is meant by *navigational*, *informational*, and *transformational search*?
- 27.5. What are the two main modes of interaction with an IR system? Describe and provide examples.
- 27.6. Explain the main differences between the database and IR systems mentioned in Table 27.1.
- 27.7. Describe the main components of the IR system as shown in Figure 27.1.
- 27.8. What are digital libraries? What types of data are typically found in them?
- 27.9. Name some digital libraries that you have accessed. What do they contain and how far back does the data go?
- 27.10. Give a brief history of IR and mention the landmark developments in this field.
- 27.11. What is the Boolean model of IR? What are its limitations?
- 27.12. What is the vector space model of IR? How does a vector get constructed to represent a document?
- 27.13. Define the TF-IDF scheme of determining the weight of a keyword in a document. Why is it necessary to include IDF in the weight of a term?
- 27.14. What are probabilistic and semantic models of IR?
- 27.15. Define *recall* and *precision* in IR systems.
- 27.16. Give the definition of *precision* and *recall* in a ranked list of results at position  $i$ .
- 27.17. How is an  $F$ -score defined as a metric of information retrieval? In what way does it account for both precision and recall?
- 27.18. What are the different types of queries in an IR system? Describe each with an example.
- 27.19. What are the approaches to processing phrase and proximity queries?
- 27.20. Describe the detailed IR process shown in Figure 27.2.
- 27.21. What is stopword removal and stemming? Why are these processes necessary for better information retrieval?
- 27.22. What is a thesaurus? How is it beneficial to IR?
- 27.23. What is information extraction? What are the different types of information extraction from structured text?
- 27.24. What are vocabularies in IR systems? What role do they play in the indexing of documents?
- 27.25. Gather five documents that contain about three sentences each and each contain some related content. Construct an inverted index of all important stems (keywords) from these documents.



- 27.26. Describe the process of constructing the result of a search request using an inverted index.
- 27.27. Define *relevance feedback*.
- 27.28. Describe the three types of Web analyses discussed in this chapter.
- 27.29. List the important tasks mentioned that are involved in analyzing Web content. Describe each in a couple of sentences.
- 27.30. What are the three categories of agent-based Web content analyses mentioned in this chapter?
- 27.31. What is the database-based approach to analyzing Web content? What are Web query systems?
- 27.32. What algorithms are popular in ranking or determining the importance of Web pages? Which algorithm was proposed by the founders of Google?
- 27.33. What is the basic idea behind the PageRank algorithm?
- 27.34. What are hubs and authority pages? How does the HITS algorithm use these concepts?
- 27.35. What can you learn from Web usage analysis? What data does it generate?
- 27.36. What mining operations are commonly performed on Web usage data? Give an example of each.
- 27.37. What are the applications of Web usage mining?
- 27.38. What is search relevance? How is it determined?
- 27.39. Define *faceted search*. Make up a set of facets for a database containing all types of buildings. For example, two facets could be “building value or price” and “building type (residential, office, warehouse, factory, and so on)”.
- 27.40. What is social search? What does collaborative social search involve?
- 27.41. Define and explain *conversational search*.
- 27.42. Define *topic modeling*.
- 27.43. How do question answering systems work?

## Selected Bibliography

Information retrieval and search technologies are active areas of research and development in industry and academia. There are many IR textbooks that provide detailed discussion of the materials that we have briefly introduced in this chapter. The book entitled *Search Engines: Information Retrieval in Practice* by Croft, Metzler, and Strohman (2009) gives a practical overview of search engine concepts and principles. *Introduction to Information Retrieval* by Manning, Raghavan, and Schütze (2008) is an authoritative book on information retrieval. Another introductory

textbook in IR is *Modern Information Retrieval* by Ricardo Baeza-Yates and Berthier Ribeiro-Neto (1999), which provides detailed coverage of various aspects of IR technology. Gerald Salton's (1968) and van Rijsbergen's (1979) classic books on information retrieval provide excellent descriptions of the foundational research done in the IR field until the late 1960s. Salton also introduced the vector space model as a model of IR. Manning and Schutze (1999) provide a good summary of natural language technologies and text preprocessing. "Interactive Information Retrieval in Digital Environments" by Xie (2008) provides a good human-centered approach to information retrieval. The book *Managing Gigabytes* by Witten, Moffat, and Bell (1999) provides detailed discussions for indexing techniques. The TREC book by Voorhees and Harman (2005) provides a description of test collection and evaluation procedures in the context of TREC competitions.

Broder (2002) classifies Web queries into three distinct classes—navigational, informational, and transactional—and presents a detailed taxonomy of Web search. Covi and Kling (1996) give a broad definition of digital libraries and discuss organizational dimensions of effective digital library use. Luhn (1957) did seminal work in IR at IBM in the 1950s on autoindexing and business intelligence. The SMART system (Salton et al. (1993)), developed at Cornell, was one of the earliest advanced IR systems that used fully automatic term indexing, hierarchical clustering, and document ranking by degree of similarity to the query. The SMART system represented documents and queries as weighted term vectors according to the vector space model.

Porter (1980) is credited with the weak and strong stemming algorithms that have become standards. Robertson (1997) developed a sophisticated weighting scheme in the City University of London Okapi system that became very popular in TREC competitions. Lenat (1995) started the Cyc project in the 1980s for incorporating formal logic and knowledge bases in information processing systems. Efforts toward creating the WordNet thesaurus continued in the 1990s and are still ongoing. WordNet concepts and principles are described in the book by Fellbaum (1998). Rocchio (1971) describes the relevance feedback algorithm, which is described in Salton's (1971) book on *The SMART Retrieval System—Experiments in Automatic Document Processing*.

Abiteboul, Buneman, and Suci (1999) provide an extensive discussion of data on the Web in their book that emphasizes semistructured data. Atzeni and Mendelzon (2000) wrote an editorial in the VLDB journal on databases and the Web. Atzeni et al. (2002) propose models and transformations for Web-based data. Abiteboul et al. (1997) propose the Lord query language for managing semistructured data.

Chakrabarti (2002) is an excellent book on knowledge discovery from the Web. The book by Liu (2006) consists of several parts, each providing a comprehensive overview of the concepts involved with Web data analysis and its applications. Excellent survey articles on Web analysis include Kosala and Blockeel (2000) and Liu et al. (2004). Etzioni (1996) provides a good starting point for understanding Web mining and describes the tasks and issues related to data mining on the World Wide Web. An excellent overview of the research issues, techniques, and development

efforts associated with Web content and usage analysis is presented by Cooley et al. (1997). Cooley (2003) focuses on mining Web usage patterns through the use of Web structure. Spiliopoulou (2000) describes Web usage analysis in detail. Web mining based on page structure is described in Madria et al. (1999) and Chakraborti et al. (1999). Algorithms to compute the rank of a Web page are given by Page et al. (1999), who describe the famous PageRank algorithm, and Kleinberg (1998), who presents the HITS algorithm.

Harth, Hose, and Schenkel (2014) present techniques for querying and managing linked data on the Web and show the potential of these techniques for research and commercial applications. Question answering technology is described in some detail by Ferrucci et al. (2010), who developed the IBM Watson system. Bikel and Zitouni (2012) is a comprehensive guide for developing robust and accurate multilingual NLP (natural language processing) systems. Blei, Ng, and Jordan (2003) provide an overview on topic modeling and latent Dirichlet allocation. For an in-depth, hands-on guide to Lucene and Solr technologies, refer to the upcoming book by Moczar (2015).

## Data Mining Concepts

Over the last several decades, many organizations have generated a large amount of machine-readable data in the form of files and databases. Existing database technology can process this data and supports query languages like SQL. However, SQL is a structured language that assumes the user is aware of the database schema. SQL supports operations of relational algebra that allow a user to select rows and columns of data from tables or join related information from tables based on common fields. In the next chapter, we will see that *data warehousing technology* affords several types of functionality: that of consolidation, aggregation, and summarization of data. Data warehouses let us view the same information along multiple dimensions. In this chapter, we will focus our attention on another very popular area of interest known as data mining. As the term connotes, **data mining** refers to the mining or discovery of new information in terms of patterns or rules from vast amounts of data. To be practically useful, data mining must be carried out efficiently on large files and databases. Although some data mining features are being provided in RDBMSs, data mining is *not* well-integrated with database management systems. The business world is presently fascinated by the potential of data mining, and the field of data mining is popularly called **business intelligence** or **data analytics**.

We will briefly review the basic concepts and principles of the extensive field of data mining, which uses techniques from such areas as machine learning, statistics, neural networks, and genetic algorithms. We will highlight the nature of the information that is discovered, the types of problems faced when trying to mine databases, and the applications of data mining. We will also survey the state of the art of a large number of commercial data mining tools (see Section 28.7) and describe a number of research advances that are needed to make this area viable.

## 28.1 Overview of Data Mining Technology

In reports such as the popular Gartner Report,<sup>1</sup> data mining has been hailed as one of the top technologies for the near future. In this section, we relate data mining to the broader area called *knowledge discovery* and contrast the two by means of an illustrative example.

### 28.1.1 Data Mining versus Data Warehousing

The goal of a data warehouse (see Chapter 29) is to support decision making with data. Data mining can be used in conjunction with a data warehouse to help with certain types of decisions. Data mining can be applied to operational databases with individual transactions. To make data mining more efficient, the data warehouse should have an aggregated or summarized collection of data. Data mining helps in extracting meaningful new patterns that cannot necessarily be found by merely querying or processing data or meta-data in the data warehouse. Therefore, data mining applications should be strongly considered early, during the design of a data warehouse. Also, data mining tools should be designed to facilitate their use in conjunction with data warehouses. In fact, for very large databases running into terabytes and even petabytes of data, successful use of data mining applications will depend first on the construction of a data warehouse.

### 28.1.2 Data Mining as a Part of the Knowledge Discovery Process

**Knowledge discovery in databases**, frequently abbreviated as **KDD**, typically encompasses more than data mining. The knowledge discovery process comprises six phases:<sup>2</sup> data selection, data cleansing, enrichment, data transformation or encoding, data mining, and the reporting and display of the discovered information.

As an example, consider a transaction database maintained by a specialty consumer goods retailer. Suppose the client data includes a customer name, zip code, phone number, date of purchase, item code, price, quantity, and total amount. A variety of new knowledge can be discovered by KDD processing on this client database. During *data selection*, data about specific items or categories of items, or from stores in a specific region or area of the country, may be selected. The *data cleansing* process then may correct invalid zip codes or eliminate records with incorrect phone prefixes. *Enrichment* typically enhances the data with additional sources of information. For example, given the client names and phone numbers, the store may purchase other data about age, income, and credit rating and append them to each record. *Data transformation* and encoding may be done to reduce the amount of

---

<sup>1</sup>The Gartner Report is one example of the many technology survey publications that corporate managers rely on to discuss and select data mining technology.

<sup>2</sup>This discussion is largely based on Adriaans and Zantinge (1996).

data. For instance, item codes may be grouped in terms of product categories into audio, video, supplies, electronic gadgets, camera, accessories, and so on. Zip codes may be aggregated into geographic regions, incomes may be divided into ranges, and so on. In Figure 29.1, we will show a process called extraction, transformation, and load (ETL) as a precursor to the data warehouse creation. If data mining is based on an existing warehouse for this retail store chain, we would expect that the cleaning has already been applied. It is only after such preprocessing that *data mining* techniques are used to mine different rules and patterns.

The result of mining may be to discover the following types of *new* information:

- **Association rules**—for example, whenever a customer buys video equipment, he or she also buys another electronic gadget.
- **Sequential patterns**—for example, suppose a customer buys a camera, and within three months he or she buys photographic supplies, then within six months he is likely to buy an accessory item. This defines a sequential pattern of transactions. A customer who buys more than twice in lean periods may be likely to buy at least once during the December holiday shopping period.
- **Classification trees**—for example, customers may be classified by frequency of visits, types of financing used, amount of purchase, or affinity for types of items; some revealing statistics may be generated for such classes.

As this retail store example shows, data mining must be preceded by significant data preparation before it can yield useful information that can directly influence business decisions.

The results of data mining may be reported in a variety of formats, such as listings, graphic outputs, summary tables, and visualizations.

### 28.1.3 Goals of Data Mining and Knowledge Discovery

Data mining is typically carried out with some end goals or applications. Broadly speaking, these goals fall into the following classes: prediction, identification, classification, and optimization.

- **Prediction.** Data mining can show how certain attributes within the data will behave in the future. Examples of predictive data mining include the analysis of buying transactions to predict what consumers will buy under certain discounts, how much sales volume a store will generate in a given period, and whether deleting a product line will yield more profits. In such applications, business logic is used coupled with data mining. In a scientific context, certain seismic wave patterns may predict an earthquake with high probability.
- **Identification.** Data patterns can be used to identify the existence of an item, an event, or an activity. For example, intruders trying to break a system may be identified by the programs executed, files accessed, and CPU time per session. In biological applications, existence of a gene may be

identified by certain sequences of nucleotide symbols in the DNA sequence. The area known as *authentication* is a form of identification. It ascertains whether a user is indeed a specific user or one from an authorized class, and it involves comparing parameters or images or signals against a database.

- **Classification.** Data mining can partition the data so that different classes or categories can be identified based on combinations of parameters. For example, customers in a supermarket can be categorized into discount-seeking shoppers, shoppers in a rush, loyal regular shoppers, shoppers attached to name brands, and infrequent shoppers. This classification may be used in different analyses of customer buying transactions as a post-mining activity. Sometimes classification based on common domain knowledge is used as an input to decompose the mining problem and make it simpler. For instance, health foods, party foods, and school lunch foods are distinct categories in the supermarket business. It makes sense to analyze relationships within and across categories as separate problems. Such categorization may be used to encode the data appropriately before subjecting it to further data mining.
- **Optimization.** One eventual goal of data mining may be to optimize the use of limited resources such as time, space, money, or materials and to maximize output variables such as sales or profits under a given set of constraints. As such, this goal of data mining resembles the objective function used in operations research problems that deals with optimization under constraints.

The term *data mining* is popularly used in a broad sense. In some situations, it includes statistical analysis and constrained optimization as well as machine learning. There is no sharp line separating data mining from these disciplines. It is beyond our scope, therefore, to discuss in detail the entire range of applications that make up this vast body of work. For a detailed understanding of the topic, readers are referred to specialized books devoted to data mining.

#### 28.1.4 Types of Knowledge Discovered during Data Mining

The term *knowledge* is broadly interpreted as involving some degree of intelligence. There is a progression from raw data to information to knowledge as we go through additional processing. Knowledge is often classified as inductive versus deductive. **Deductive knowledge** deduces new information based on applying *prespecified* logical rules of deduction on the given data. Data mining addresses **inductive knowledge**, which discovers new rules and patterns from the supplied data. Knowledge can be represented in many forms: In an unstructured sense, it can be represented by rules or propositional logic. In a structured form, it may be represented in decision trees, semantic networks, neural networks, or hierarchies of classes or frames. It is common to describe the knowledge discovered during data mining as follows:

- **Association rules.** These rules correlate the presence of a set of items with another range of values for another set of variables. Examples: (1) When a female retail shopper buys a handbag, she is likely to buy shoes. (2) An X-ray image containing characteristics a and b is likely to also exhibit characteristic c.



- **Classification hierarchies.** The goal is to work from an existing set of events or transactions to create a hierarchy of classes. Examples: (1) A population may be divided into five ranges of credit worthiness based on a history of previous credit transactions. (2) A model may be developed for the factors that determine the desirability of a store location on a 1–10 scale. (3) Mutual funds may be classified based on performance data using characteristics such as growth, income, and stability.
- **Sequential patterns.** A sequence of actions or events is sought. Example: If a patient underwent cardiac bypass surgery for blocked arteries and an aneurysm and later developed high blood urea within a year of surgery, he or she is likely to suffer from kidney failure within the next 18 months. Detecting sequential patterns is equivalent to detecting associations among events with certain temporal relationships.
- **Patterns within time series.** Similarities can be detected within positions of a **time series** of data, which is a sequence of data taken at regular intervals, such as daily sales or daily closing stock prices. Examples: (1) Stocks of a utility company, ABC Power, and a financial company, XYZ Securities, showed the same pattern during 2014 in terms of closing stock prices. (2) Two products show the same selling pattern in summer but a different one in winter. (3) A pattern in solar magnetic wind may be used to predict changes in Earth’s atmospheric conditions.
- **Clustering.** A given population of events or items can be partitioned (segmented) into sets of “similar” elements. Examples: (1) An entire population of treatment data on a disease may be divided into groups based on the similarity of side effects produced. (2) The adult population in the United States may be categorized into five groups from *most likely to buy* to *least likely to buy* a new product. (3) The Web accesses made by a collection of users against a set of documents (say, in a digital library) may be analyzed in terms of the keywords of documents to reveal clusters or categories of users.

For most applications, the desired knowledge is a combination of the above types. We expand on each of the above knowledge types in the following sections.

## 28.2 Association Rules

### 28.2.1 Market-Basket Model, Support, and Confidence

One of the major technologies in data mining involves the discovery of association rules. The database is regarded as a collection of transactions, each involving a set of items. A common example is that of **market-basket data**. Here the market basket corresponds to the sets of items a consumer buys in a supermarket during one visit. Consider four such transactions in a random sample shown in Figure 28.1.

An **association rule** is of the form  $X \Rightarrow Y$ , where  $X = \{x_1, x_2, \dots, x_n\}$ , and  $Y = \{y_1, y_2, \dots, y_m\}$  are sets of items, with  $x_i$  and  $y_j$  being distinct items for all  $i$  and all  $j$ . This

association states that if a customer buys  $X$ , he or she is also likely to buy  $Y$ . In general, any association rule has the form LHS (left-hand side)  $\Rightarrow$  RHS (right-hand side), where LHS and RHS are sets of items. The set  $LHS \cup RHS$  is called an **itemset**, the set of items purchased by customers. For an association rule to be of interest to a data miner, the rule should satisfy some interest measure. Two common interest measures are support and confidence.

The **support** for a rule  $LHS \Rightarrow RHS$  is with respect to the itemset; it refers to how frequently a specific itemset occurs in the database. That is, the support is the percentage of transactions that contain all of the items in the itemset  $LHS \cup RHS$ . If the support is low, it implies that there is no overwhelming evidence that items in  $LHS \cup RHS$  occur together because the itemset occurs in only a small fraction of transactions. Another term for support is *prevalence* of the rule.

The **confidence** is with regard to the implication shown in the rule. The confidence of the rule  $LHS \Rightarrow RHS$  is computed as the  $\text{support}(LHS \cup RHS) / \text{support}(LHS)$ . We can think of it as the probability that the items in RHS will be purchased given that the items in LHS are purchased by a customer. Another term for confidence is *strength* of the rule.

As an example of support and confidence, consider the following two rules:  $\text{milk} \Rightarrow \text{juice}$  and  $\text{bread} \Rightarrow \text{juice}$ . Looking at our four sample transactions in Figure 28.1, we see that the support of  $\{\text{milk}, \text{juice}\}$  is 50% and the support of  $\{\text{bread}, \text{juice}\}$  is only 25%. The confidence of  $\text{milk} \Rightarrow \text{juice}$  is 66.7% (meaning that, of three transactions in which milk occurs, two contain juice) and the confidence of  $\text{bread} \Rightarrow \text{juice}$  is 50% (meaning that one of two transactions containing bread also contains juice).

As we can see, support and confidence do not necessarily go hand in hand. The goal of mining association rules, then, is to generate all possible rules that exceed some minimum user-specified support and confidence thresholds. The problem is thus decomposed into two subproblems:

1. Generate all itemsets that have a support that exceeds the threshold. These sets of items are called **large** (or **frequent**) **itemsets**. Note that large here means large support.
2. For each large itemset, all the rules that have a minimum confidence are generated as follows: For a large itemset  $X$  and  $Y \subset X$ , let  $Z = X - Y$ ; then if  $\text{support}(X) / \text{support}(Z) > \text{minimum confidence}$ , the rule  $Z \Rightarrow Y$  (that is,  $X - Y \Rightarrow Y$ ) is a valid rule.

Generating rules by using all large itemsets and their supports is relatively straightforward. However, discovering all large itemsets together with the value for their

**Figure 28.1**

Sample transactions in market-basket model.

Transaction_id	Time	Items_bought
101	6:35	milk, bread, cookies, juice
792	7:38	milk, juice
1130	8:05	milk, eggs
1735	8:40	bread, cookies, coffee

support is a major problem if the cardinality of the set of items is very high. A typical supermarket has thousands of items. The number of distinct itemsets is  $2^m$ , where  $m$  is the number of items, and counting support for all possible itemsets becomes very computation intensive. To reduce the combinatorial search space, algorithms for finding association rules utilize the following properties:

- A subset of a large itemset must also be large (that is, each subset of a large itemset exceeds the minimum required support).
- Conversely, a superset of a small itemset is also small (implying that it does not have enough support).

The first property is referred to as **downward closure**. The second property, called the **antimonotonicity** property, helps to reduce the search space of possible solutions. That is, once an itemset is found to be small (not a large itemset), then any extension to that itemset, formed by adding one or more items to the set, will also yield a small itemset.

### 28.2.2 Apriori Algorithm

The first algorithm to use the downward closure and antimonotonicity properties was the **apriori algorithm**, shown as Algorithm 28.1.

We illustrate Algorithm 28.1 using the transaction data in Figure 28.1 using a minimum support of 0.5. The candidate 1-itemsets are {milk, bread, juice, cookies, eggs, coffee} and their respective supports are 0.75, 0.5, 0.5, 0.5, 0.25, and 0.25. The first four items qualify for  $L_1$  since each support is greater than or equal to 0.5. In the first iteration of the repeat-loop, we extend the frequent 1-itemsets to create the candidate frequent 2-itemsets,  $C_2$ .  $C_2$  contains {milk, bread}, {milk, juice}, {bread, juice}, {milk, cookies}, {bread, cookies}, and {juice, cookies}. Notice, for example, that {milk, eggs} does not appear in  $C_2$  since {eggs} is small (by the antimonotonicity property) and does not appear in  $L_1$ . The supports for the six sets contained in  $C_2$  are 0.25, 0.5, 0.25, 0.25, 0.5, and 0.25 and are computed by scanning the set of transactions. Only the second 2-itemset {milk, juice} and the fifth 2-itemset {bread, cookies} have support greater than or equal to 0.5. These two 2-itemsets form the frequent 2-itemsets,  $L_2$ .

#### Algorithm 28.1. Apriori Algorithm for Finding Frequent (Large) Itemsets

**Input:** Database of  $m$  transactions,  $D$ , and a minimum support,  $mins$ , represented as a fraction of  $m$ .

**Output:** Frequent itemsets,  $L_1, L_2, \dots, L_k$

**Begin** /\* steps or statements are numbered for better readability \*/

1. Compute  $\text{support}(i_j) = \text{count}(i_j)/m$  for each individual item,  $i_1, i_2, \dots, i_n$  by scanning the database once and counting the number of transactions that item  $i_j$  appears in (that is,  $\text{count}(i_j)$ );
2. The candidate frequent 1-itemset,  $C_1$ , will be the set of items  $i_1, i_2, \dots, i_n$ ;

3. The subset of items containing  $i_j$  from  $C_1$  where  $\text{support}(i_j) \geq \text{mins}$  becomes the frequent 1-itemset,  $L_1$ ;
  4.  $k = 1$ ;  
     termination = false;  
     **repeat**
    1.  $L_{k+1} = (\text{empty set})$ ;
    2. Create the candidate frequent  $(k+1)$ -itemset,  $C_{k+1}$ , by combining members of  $L_k$  that have  $k-1$  items in common (this forms candidate frequent  $(k+1)$ -itemsets by selectively extending frequent  $k$ -itemsets by one item);
    3. In addition, only consider as elements of  $C_{k+1}$  those  $k+1$  items such that every subset of size  $k$  appears in  $L_k$ ;
    4. Scan the database once and compute the support for each member of  $C_{k+1}$ ; if the support for a member of  $C_{k+1} \geq \text{mins}$  then add that member to  $L_{k+1}$ ;
    5. If  $L_{k+1}$  is empty then termination = true  
     else  $k = k + 1$ ;**until termination**;
- End**;

In the next iteration of the repeat-loop, we construct candidate frequent 3-itemsets by adding additional items to sets in  $L_2$ . However, for no extension of itemsets in  $L_2$  will all 2-item subsets be contained in  $L_2$ . For example, consider {milk, juice, bread}; the 2-itemset {milk, bread} is not in  $L_2$ , hence {milk, juice, bread} cannot be a frequent 3-itemset by the downward closure property. At this point the algorithm terminates with  $L_1$  equal to {{milk}, {bread}, {juice}, {cookies}} and  $L_2$  equal to {{milk, juice}, {bread, cookies}}.

Several other algorithms have been proposed to mine association rules. They vary mainly in terms of how the candidate itemsets are generated and how the supports for the candidate itemsets are counted. Some algorithms use data structures such as bitmaps and hashtrees to keep information about itemsets. Several algorithms have been proposed that use multiple scans of the database because the potential number of itemsets,  $2^m$ , can be too large to set up counters during a single scan. We will examine three improved algorithms (compared to the Apriori algorithm) for association rule mining: the sampling algorithm, the frequent-pattern tree algorithm, and the partition algorithm.

### 28.2.3 Sampling Algorithm

The main idea for the **sampling algorithm** is to select a small sample, one that fits in main memory, of the database of transactions and to determine the frequent itemsets from that sample. If those frequent itemsets form a superset of the frequent itemsets for the entire database, then we can determine the real frequent itemsets by scanning the remainder of the database in order to compute the exact support values for the superset itemsets. A superset of the frequent itemsets can usually be

found from the sample by using, for example, the apriori algorithm, with a lowered minimum support.

In rare cases, some frequent itemsets may be missed and a second scan of the database is needed. To decide whether any frequent itemsets have been missed, the concept of the *negative border* is used. The negative border with respect to a frequent itemset,  $S$ , and set of items,  $I$ , is the minimal itemsets contained in  $\text{PowerSet}(I)$  and not in  $S$ . The basic idea is that the negative border of a set of frequent itemsets contains the closest itemsets that could also be frequent. Consider the case where a set  $X$  is not contained in the frequent itemsets. If all subsets of  $X$  are contained in the set of frequent itemsets, then  $X$  would be in the negative border.

We illustrate this with the following example. Consider the set of items  $I = \{A, B, C, D, E\}$  and let the combined frequent itemsets of size 1 to 3 be  $S = \{\{A\}, \{B\}, \{C\}, \{D\}, \{AB\}, \{AC\}, \{BC\}, \{AD\}, \{CD\}, \{ABC\}\}$ . The negative border is  $\{\{E\}, \{BD\}, \{ACD\}\}$ . The set  $\{E\}$  is the only 1-itemset not contained in  $S$ ,  $\{BD\}$  is the only 2-itemset not in  $S$  but whose 1-itemset subsets are, and  $\{ACD\}$  is the only 3-itemset whose 2-itemset subsets are all in  $S$ . The negative border is important since it is necessary to determine the support for those itemsets in the negative border to ensure that no large itemsets are missed from analyzing the sample data.

Support for the negative border is determined when the remainder of the database is scanned. If we find that an itemset,  $X$ , in the negative border belongs in the set of all frequent itemsets, then there is a potential for a superset of  $X$  to also be frequent. If this happens, then a second pass over the database is needed to make sure that all frequent itemsets are found.

### 28.2.4 Frequent-Pattern (FP) Tree and FP-Growth Algorithm

The **frequent-pattern tree (FP-tree)** is motivated by the fact that apriori-based algorithms may generate and test a very large number of candidate itemsets. For example, with 1,000 frequent 1-itemsets, the apriori algorithm would have to generate

$$\binom{1000}{2}$$

or 499,500 candidate 2-itemsets. The **FP-growth algorithm** is one approach that eliminates the generation of a large number of candidate itemsets.

The algorithm first produces a compressed version of the database in terms of an FP-tree (frequent-pattern tree). The FP-tree stores relevant itemset information and allows for the efficient discovery of frequent itemsets. The actual mining process adopts a divide-and-conquer strategy, where the mining process is decomposed into a set of smaller tasks that each operates on a conditional FP-tree, a subset (projection) of the original tree. To start with, we examine how the FP-tree is constructed. The database is first scanned and the frequent 1-itemsets along with their support are computed. With this algorithm, the support is the *count* of transactions

containing the item rather than the fraction of transactions containing the item. The frequent 1-itemsets are then sorted in nonincreasing order of their support. Next, the root of the FP-tree is created with a NULL label. The database is scanned a second time and for each transaction  $T$  in the database, the frequent 1-itemsets in  $T$  are placed in order as was done with the frequent 1-itemsets. We can designate this sorted list for  $T$  as consisting of a first item, the head, and the remaining items, the tail. The itemset information (head, tail) is inserted into the FP-tree recursively, starting at the root node, as follows:

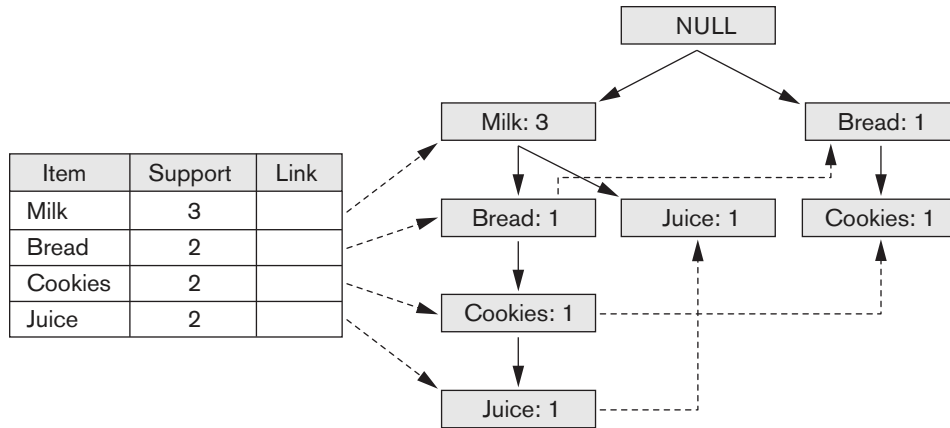
1. If the current node,  $N$ , of the FP-tree has a child with an item name = head, then increment the count associated with node  $N$  by 1, else create a new node,  $N$ , with a count of 1, link  $N$  to its parent and link  $N$  with the item header table (used for efficient tree traversal).
2. If the tail is nonempty, then repeat step (1) using as the sorted list only the tail, that is, the old head is removed and the new head is the first item from the tail and the remaining items become the new tail.

The item header table, created during the process of building the FP-tree, contains three fields per entry for each frequent item: item identifier, support count, and node link. The item identifier and support count are self-explanatory. The node link is a pointer to an occurrence of that item in the FP-tree. Since multiple occurrences of a single item may appear in the FP-tree, these items are linked together as a list where the start of the list is pointed to by the node link in the item header table. We illustrate the building of the FP-tree using the transaction data in Figure 28.1. Let us use a minimum support of 2. One pass over the four transactions yields the following frequent 1-itemsets with associated support:  $\{(milk, 3)\}$ ,  $\{(bread, 2)\}$ ,  $\{(cookies, 2)\}$ ,  $\{(juice, 2)\}$ . The database is scanned a second time and each transaction will be processed again.

For the first transaction, we create the sorted list,  $T = \{milk, bread, cookies, juice\}$ . The items in  $T$  are the frequent 1-itemsets from the first transaction. The items are ordered based on the nonincreasing ordering of the count of the 1-itemsets found in pass 1 (that is, milk first, bread second, and so on). We create a NULL root node for the FP-tree and insert *milk* as a child of the root, *bread* as a child of *milk*, *cookies* as a child of *bread*, and *juice* as a child of *cookies*. We adjust the entries for the frequent items in the item header table.

For the second transaction, we have the sorted list  $\{milk, juice\}$ . Starting at the root, we see that a child node with label *milk* exists, so we move to that node and update its count (to account for the second transaction that contains milk). We see that there is no child of the current node with label *juice*, so we create a new node with label *juice*. The item header table is adjusted.

The third transaction only has 1-frequent item,  $\{milk\}$ . Again, starting at the root, we see that the node with label *milk* exists, so we move to that node, increment its count, and adjust the item header table. The final transaction contains frequent items,  $\{bread, cookies\}$ . At the root node, we see that a child with label *bread* does not exist. Thus, we create a new child of the root, initialize its counter, and then



**Figure 28.2**  
FP-tree and item  
header table.

insert *cookies* as a child of this node and initialize its count. After the item header table is updated, we end up with the FP-tree and item header table as shown in Figure 28.2. If we examine this FP-tree, we see that it indeed represents the original transactions in a compressed format (that is, only showing the items from each transaction that are large 1-itemsets).

Algorithm 28.2 is used for mining the FP-tree for frequent patterns. With the FP-tree, it is possible to find all frequent patterns that contain a given frequent item by starting from the item header table for that item and traversing the node links in the FP-tree. The algorithm starts with a frequent 1-itemset (suffix pattern) and constructs its conditional pattern base and then its conditional FP-tree. The conditional pattern base is made up of a set of prefix paths, that is, where the frequent item is a suffix. For example, if we consider the item juice, we see from Figure 28.2 that there are two paths in the FP-tree that end with juice: (milk, bread, cookies, juice) and (milk, juice). The two associated prefix paths are (milk, bread, cookies) and (milk). The conditional FP-tree is constructed from the patterns in the conditional pattern base. The mining is recursively performed on this FP-tree. The frequent patterns are formed by concatenating the suffix pattern with the frequent patterns produced from a conditional FP-tree.

**Algorithm 28.2.** FP-Growth Algorithm for Finding Frequent Itemsets

**Input:** FP-tree and a minimum support, mins

**Output:** frequent patterns (itemsets)  
procedure FP-growth (tree, alpha);

**Begin**

if tree contains a single path  $P$  then  
for each combination, beta, of the nodes in the path  
generate pattern ( $\text{beta} \cup \text{alpha}$ )  
with support = minimum support of nodes in beta



```

else
    for each item,  $i$ , in the header of the tree do
        begin
            generate pattern  $\beta = (i \cup \alpha)$  with support =  $i$ .support;
            construct  $\beta$ 's conditional pattern base;
            construct  $\beta$ 's conditional FP-tree,  $\beta\_tree$ ;
            if  $\beta\_tree$  is not empty then
                FP-growth( $\beta\_tree$ ,  $\beta$ );
        end;
    End;

```

We illustrate the algorithm using the data in Figure 28.1 and the tree in Figure 28.2. The procedure FP-growth is called with the two parameters: the original FP-tree and NULL for the variable  $\alpha$ . Since the original FP-tree has more than a single path, we execute the else part of the first if statement. We start with the frequent item, juice. We will examine the frequent items in order of lowest support (that is, from the last entry in the table to the first). The variable  $\beta$  is set to juice with support equal to 2.

Following the node link in the item header table, we construct the conditional pattern base consisting of two paths (with juice as suffix). These are (milk, bread, cookies: 1) and (milk: 1). The conditional FP-tree consists of only a single node, milk: 2. This is due to a support of only 1 for node bread and cookies, which is below the minimal support of 2. The algorithm is called recursively with an FP-tree of only a single node (that is, milk: 2) and a  $\beta$  value of juice. Since this FP-tree only has one path, all combinations of  $\beta$  and nodes in the path are generated—that is, {milk, juice}—with support of 2.

Next, the frequent item, cookies, is used. The variable  $\beta$  is set to cookies with support = 2. Following the node link in the item header table, we construct the conditional pattern base consisting of two paths. These are (milk, bread: 1) and (bread: 1). The conditional FP-tree is only a single node, bread: 2. The algorithm is called recursively with an FP-tree of only a single node (that is, bread: 2) and a  $\beta$  value of cookies. Since this FP-tree only has one path, all combinations of  $\beta$  and nodes in the path are generated, that is, {bread, cookies} with support of 2. The frequent item, bread, is considered next. The variable  $\beta$  is set to bread with support = 2. Following the node link in the item header table, we construct the conditional pattern base consisting of one path, which is (milk: 1). The conditional FP-tree is empty since the count is less than the minimum support. Since the conditional FP-tree is empty, no frequent patterns will be generated.

The last frequent item to consider is milk. This is the top item in the item header table and as such has an empty conditional pattern base and empty conditional FP-tree. As a result, no frequent patterns are added. The result of executing the algorithm is the following frequent patterns (or itemsets) with their support: {{milk: 3}, {bread: 2}, {cookies: 2}, {juice: 2}, {milk, juice: 2}, {bread, cookies: 2}}.

### 28.2.5 Partition Algorithm

Another algorithm, called the **partition algorithm**,<sup>3</sup> is summarized below. If we are given a database with a small number of potential large itemsets, say, a few thousand, then the support for all of them can be tested in one scan by using a partitioning technique. Partitioning divides the database into nonoverlapping subsets; these are individually considered as separate databases and all large itemsets for that partition, called *local frequent itemsets*, are generated in one pass. The apriori algorithm can then be used efficiently on each partition if it fits entirely in main memory. Partitions are chosen in such a way that each partition can be accommodated in main memory. As such, a partition is read only once in each pass. The only caveat with the partition method is that the minimum support used for each partition has a slightly different meaning from the original value. The minimum support is based on the size of the partition rather than the size of the database for determining local frequent (large) itemsets. The actual support threshold value is the same as given earlier, but the support is computed only for a partition.

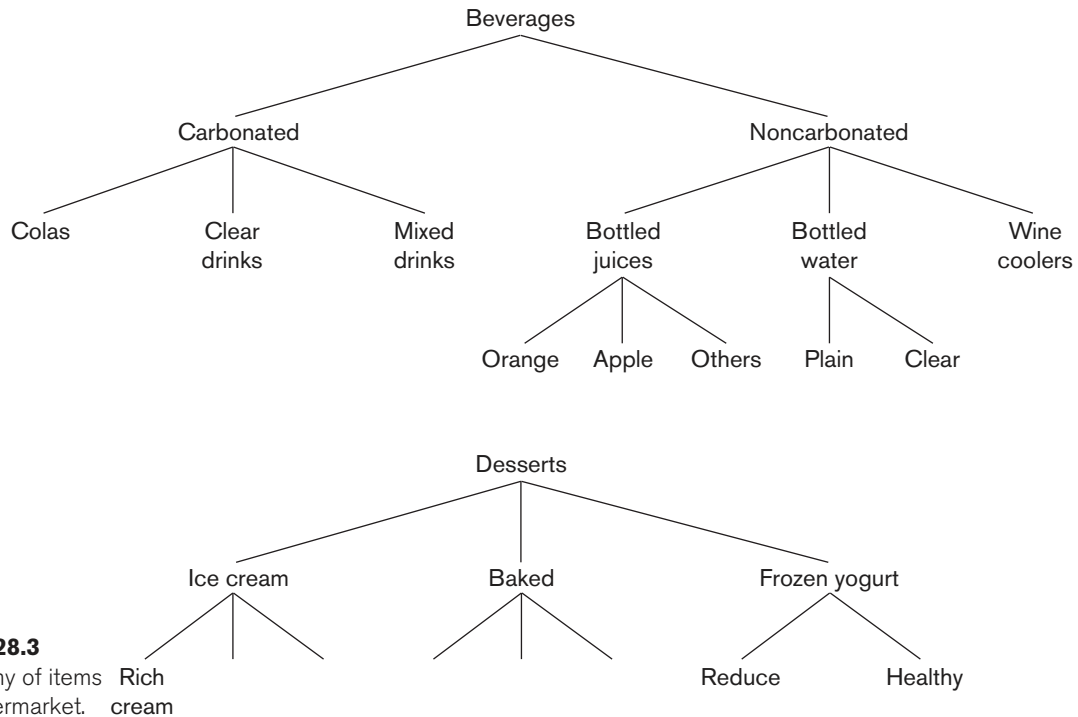
At the end of pass one, we take the union of all frequent itemsets from each partition. This forms the global candidate frequent itemsets for the entire database. When these lists are merged, they may contain some false positives. That is, some of the itemsets that are frequent (large) in one partition may not qualify in several other partitions and hence may not exceed the minimum support when the original database is considered. Note that there are no false negatives; no large itemsets will be missed. The global candidate large itemsets identified in pass one are verified in pass two; that is, their actual support is measured for the *entire* database. At the end of phase two, all global large itemsets are identified. The partition algorithm lends itself naturally to a parallel or distributed implementation for better efficiency. Further improvements to this algorithm have been suggested.<sup>4</sup>

### 28.2.6 Other Types of Association Rules

**Association Rules among Hierarchies.** There are certain types of associations that are particularly interesting for a special reason. These associations occur among hierarchies of items. Typically, it is possible to divide items among disjoint hierarchies based on the nature of the domain. For example, foods in a supermarket, items in a department store, or articles in a sports shop can be categorized into classes and subclasses that give rise to hierarchies. Consider Figure 28.3, which shows the taxonomy of items in a supermarket. The figure shows two hierarchies—beverages and desserts, respectively. The entire groups may not produce associations of the form *beverages*  $\Rightarrow$  *desserts*, or *desserts*  $\Rightarrow$  *beverages*. However, associations of the type *Healthy-brand frozen yogurt*  $\Rightarrow$  *bottled water*, or *Rich*

<sup>3</sup>See Savasere et al. (1995) for details of the algorithm, the data structures used to implement it, and its performance comparisons.

<sup>4</sup>See Cheung et al. (1996) and Lin and Dunham (1998).

**Figure 28.3**

Taxonomy of items in a supermarket.

cream-brand ice cream  $\Rightarrow$  wine cooler may produce enough confidence and support to be valid association rules of interest.

Therefore, if the application area has a natural classification of the itemsets into hierarchies, discovering associations *within* the hierarchies is of no particular interest. The ones of specific interest are associations *across* hierarchies. They may occur among item groupings at different levels.

**Multidimensional Associations.** Discovering association rules involves searching for patterns in a file. In Figure 28.1, we have an example of a file of customer transactions with three dimensions: Transaction\_id, Time, and Items\_bought. However, our data mining tasks and algorithms introduced up to this point only involve one dimension: Items\_bought. The following rule is an example of including the label of the single dimension: Items\_bought(milk)  $\Rightarrow$  Items\_bought(juice). It may be of interest to find association rules that involve multiple dimensions, for example, Time(6:30 ... 8:00)  $\Rightarrow$  Items\_bought(milk). Rules like these are called *multidimensional association rules*. The dimensions represent attributes of records of a file or, in terms of relations, columns of rows of a relation, and can be categorical or quantitative. Categorical attributes have a finite set of values that display no ordering relationship. Quantitative attributes are numeric and their values display an ordering relationship, for example,  $<$ . Items\_bought is an example of a categorical attribute and Transaction\_id and Time are quantitative.

One approach to handling a quantitative attribute is to partition its values into non-overlapping intervals that are assigned labels. This can be done in a static manner based on domain-specific knowledge. For example, a concept hierarchy may group values for Salary into three distinct classes: low income ( $0 < \text{Salary} < 29,999$ ), middle income ( $30,000 < \text{Salary} < 74,999$ ), and high income ( $\text{Salary} > 75,000$ ). From here, the typical apriori-type algorithm or one of its variants can be used for the rule mining since the quantitative attributes now look like categorical attributes. Another approach to partitioning is to group attribute values based on data distribution (for example, equi-depth partitioning) and to assign integer values to each partition. The partitioning at this stage may be relatively fine, that is, a larger number of intervals. Then during the mining process, these partitions may combine with other adjacent partitions if their support is less than some predefined maximum value. An apriori-type algorithm can be used here as well for the data mining.

**Negative Associations.** The problem of discovering a negative association is harder than that of discovering a positive association. A negative association is of the following type: *60% of customers who buy potato chips do not buy bottled water*. (Here, the 60% refers to the confidence for the negative association rule.) In a database with 10,000 items, there are 210,000 possible combinations of items, a majority of which do not appear even once in the database. If the absence of a certain item combination is taken to mean a negative association, then we potentially have millions and millions of negative association rules with RHSs that are of no interest at all. The problem, then, is to find only *interesting* negative rules. In general, we are interested in cases in which two specific sets of items appear very rarely in the same transaction. This poses two problems.

1. For a total item inventory of 10,000 items, the probability of any two being bought together is  $(1/10,000) * (1/10,000) = 10^{-8}$ . If we find the actual support for these two occurring together to be zero, that does not represent a significant departure from expectation and hence is not an interesting (negative) association.
2. The other problem is more serious. We are looking for item combinations with very low support, and there are millions and millions with low or even zero support. For example, a data set of 10 million transactions has most of the 2.5 billion pairwise combinations of 10,000 items missing. This would generate billions of useless rules.

Therefore, to make negative association rules interesting, we must use prior knowledge about the itemsets. One approach is to use hierarchies. Suppose we use the hierarchies of soft drinks and chips shown in Figure 28.4.



**Figure 28.4**  
Simple hierarchy of  
soft drinks and chips.

A strong positive association has been shown between soft drinks and chips. If we find a large support for the fact that when customers buy Days chips they predominantly buy Topsy and *not* Joke and *not* Wakeup, that would be interesting because we would normally expect that if there is a strong association between Days and Topsy, there should also be such a strong association between Days and Joke or Days and Wakeup.<sup>5</sup>

In the frozen yogurt and bottled water groupings shown in Figure 28.3, suppose the Reduce versus Healthy-brand division is 80–20 and the Plain and Clear brands division is 60–40 among respective categories. This would give a joint probability of Reduce frozen yogurt being purchased with Plain bottled water as 48% among the transactions containing a frozen yogurt and bottled water. If this support, however, is found to be only 20%, it would indicate a significant negative association among Reduce yogurt and Plain bottled water; again, that would be interesting.

The problem of finding negative association is important in the above situations given the domain knowledge in the form of item generalization hierarchies (that is, the beverage given and desserts hierarchies shown in Figure 28.3), the existing positive associations (such as between the frozen yogurt and bottled water groups), and the distribution of items (such as the name brands within related groups). The scope of discovery of negative associations is limited in terms of knowing the item hierarchies and distributions. Exponential growth of negative associations remains a challenge.

### 28.2.7 Additional Considerations for Association Rules

The mining of association rules in real-life databases is complicated by the following factors:

- The cardinality of itemsets in most situations is extremely large, and the volume of transactions is very high as well. Some operational databases in retailing and communication industries collect tens of millions of transactions per day.
- Transactions show variability in such factors as geographic location and seasons, making sampling difficult.
- Item classifications exist along multiple dimensions. Hence, driving the discovery process with domain knowledge, particularly for negative rules, is extremely difficult.
- Quality of data is variable; significant problems exist with missing, erroneous, conflicting, as well as redundant data in many industries.

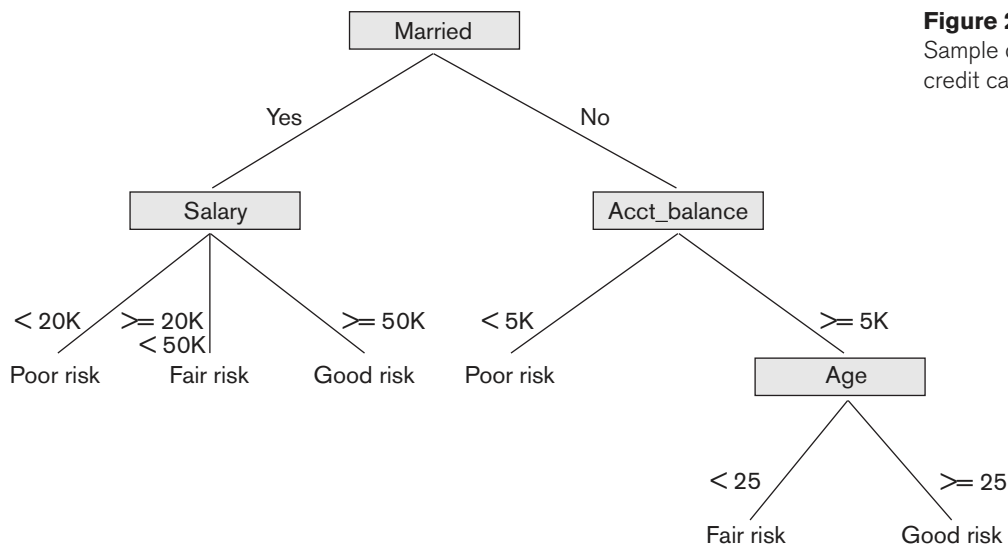
---

<sup>5</sup>For simplicity we are assuming a uniform distribution of transactions among members of a hierarchy.

## 28.3 Classification

**Classification** is the process of learning a model that describes different classes of data. The classes are predetermined. For example, in a banking application, customers who apply for a credit card may be classified as a *poor risk*, *fair risk*, or *good risk*. Hence this type of activity is also called **supervised learning**. Once the model is built, it can be used to classify new data. The first step—learning the model—is accomplished by using a training set of data that has already been classified. Each record in the training data contains an attribute, called the *class* label, which indicates which class the record belongs to. The model that is produced is usually in the form of a decision tree or a set of rules. Some of the important issues with regard to the model and the algorithm that produces the model include the model's ability to predict the correct class of new data, the computational cost associated with the algorithm, and the scalability of the algorithm.

We will examine the approach where our model is in the form of a decision tree. A **decision tree** is simply a graphical representation of the description of each class or, in other words, a representation of the classification rules. A sample decision tree is pictured in Figure 28.5. We see from Figure 28.5 that if a customer is *married* and if salary  $\geq 50K$ , then she is a good risk for a bank credit card. This is one of the rules that describe the class *good risk*. Traversing the decision tree from the root to each leaf node forms other rules for this class and the two other classes. Algorithm 28.3 shows the procedure for constructing a decision tree from a training data set. Initially, all training samples are at the root of the tree. The samples are partitioned recursively based on selected attributes. The attribute used at a node to partition the samples is the one with the best splitting criterion, for example, the one that maximizes the information gain measure.



**Figure 28.5**  
Sample decision tree for  
credit card applications.

**Algorithm 28.3.** Algorithm for Decision Tree Induction

**Input:** Set of training data records:  $R_1, R_2, \dots, R_m$  and set of attributes:  $A_1, A_2, \dots, A_n$

**Output:** Decision tree

procedure Build\_tree (records, attributes);

**Begin**

    create a node  $N$ ;

    if all records belong to the same class  $C$ , then

        return  $N$  as a leaf node with class label  $C$ ;

    if attributes is empty then

        return  $N$  as a leaf node with class label  $C$ , such that the majority of records belong to it;

    select attribute  $A_i$  (*with the highest information gain*) from attributes;

    label node  $N$  with  $A_i$ ;

    for each known value,  $v_j$ , of  $A_i$  do

**begin**

            add a branch from node  $N$  for the condition  $A_i = v_j$ ;

$S_j$  = subset of records where  $A_i = v_j$ ;

            if  $S_j$  is empty then

                add a leaf,  $L$ , with class label  $C$ , such that the majority of records belong to it and return  $L$

            else add the node returned by Build\_tree( $S_j$ , attributes –  $A_i$ );

**end;**

**End;**

Before we illustrate Algorithm 28.3, we will explain the **information gain** measure in more detail. The use of **entropy** as the information gain measure is motivated by the goal of minimizing the information needed to classify the sample data in the resulting partitions and thus minimizing the expected number of conditional tests needed to classify a new record. The expected information needed to classify training data of  $s$  samples, where the Class attribute has  $n$  values ( $v_1, \dots, v_n$ ) and  $s_i$  is the number of samples belonging to class label  $v_i$ , is given by

$$I(S_1, S_2, \dots, S_n) = - \sum_{i=1}^n p_i \log_2 p_i$$

where  $p_i$  is the probability that a random sample belongs to the class with label  $v_i$ . An estimate for  $p_i$  is  $s_i/s$ . Consider an attribute  $A$  with values  $\{v_1, \dots, v_m\}$  used as the test attribute for splitting in the decision tree. Attribute  $A$  partitions the samples into the subsets  $S_1, \dots, S_m$  where samples in each  $S_j$  have a value of  $v_j$  for attribute  $A$ . Each  $S_j$  may contain samples that belong to any of the classes. The number of samples in  $S_j$  that belong to class  $i$  can be denoted as  $s_{ij}$ . The entropy associated with using attribute  $A$  as the test attribute is defined as

$$E(A) = \sum_{j=1}^m \frac{S_{1j} + \dots + S_{nj}}{S} \times I(S_{1j}, \dots, S_{nj})$$



RID	Married	Salary	Acct_balance	Age	Loanworthy
1	no	$\geq 50K$	$< 5K$	$\geq 25$	yes
2	yes	$\geq 50K$	$\geq 5K$	$\geq 25$	yes
3	yes	20K. . . 50K	$< 5K$	$< 25$	no
4	no	$< 20K$	$\geq 5K$	$< 25$	no
5	no	$< 20K$	$< 5K$	$\geq 25$	no
6	yes	20K. . . 50K	$\geq 5K$	$\geq 25$	yes

**Figure 28.6**  
Sample training data  
for classification  
algorithm.

$I(s_{1j}, \dots, s_{nj})$  can be defined using the formulation for  $I(s_1, \dots, s_n)$  with  $p_i$  being replaced by  $p_{ij}$  where  $p_{ij} = s_{ij}/s_j$ . Now the information gain by partitioning on attribute  $A$ ,  $\text{Gain}(A)$ , is defined as  $I(s_1, \dots, s_n) - E(A)$ . We can use the sample training data from Figure 28.6 to illustrate the algorithm.

The attribute RID represents the record identifier used for identifying an individual record and is an internal attribute. We use it to identify a particular record in our example. First, we compute the expected information needed to classify the training data of 6 records as  $I(s_1, s_2)$  where there are two classes: the first class label value corresponds to *yes* and the second to *no*. So

$$I(3,3) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 1$$

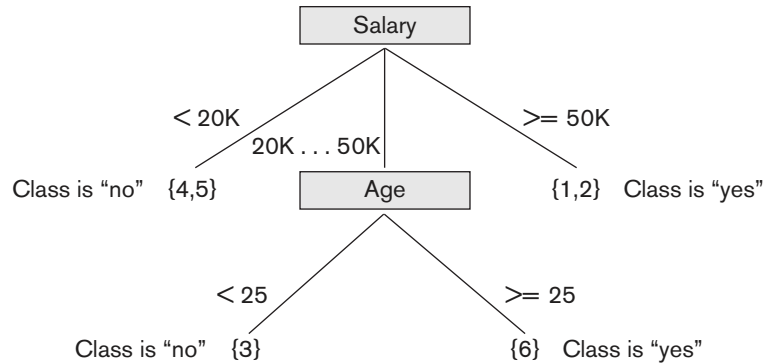
Now, we compute the entropy for each of the four attributes as shown below. For Married = yes, we have  $s_{11} = 2$ ,  $s_{21} = 1$  and  $I(s_{11}, s_{21}) = 0.92$ . For Married = no, we have  $s_{12} = 1$ ,  $s_{22} = 2$  and  $I(s_{12}, s_{22}) = 0.92$ . So the expected information needed to classify a sample using attribute Married as the partitioning attribute is

$$E(\text{Married}) = 3/6 I(s_{11}, s_{21}) + 3/6 I(s_{12}, s_{22}) = 0.92$$

The gain in information,  $\text{Gain}(\text{Married})$ , would be  $1 - 0.92 = 0.08$ . If we follow similar steps for computing the gain with respect to the other three attributes we end up with

$$\begin{array}{lll} E(\text{Salary}) = 0.33 & \text{and} & \text{Gain}(\text{Salary}) = 0.67 \\ E(\text{Acct\_balance}) = 0.92 & \text{and} & \text{Gain}(\text{Acct\_balance}) = 0.08 \\ E(\text{Age}) = 0.54 & \text{and} & \text{Gain}(\text{Age}) = 0.46 \end{array}$$

Since the greatest gain occurs for attribute Salary, it is chosen as the partitioning attribute. The root of the tree is created with label *Salary* and has three branches, one for each value of Salary. For two of the three values, that is,  $< 20K$  and  $> 50K$ , all the samples that are partitioned accordingly (records with RIDs 4 and 5 for  $< 20K$  and records with RIDs 1 and 2 for  $\geq 50K$ ) fall within the same class *loanworthy no* and *loanworthy yes*, respectively, for those two values. So we create a leaf node for each. The only branch that needs to be expanded is for the value 20K ... 50K with two samples, records with RIDs 3 and 6 in the training data. Continuing the process using these two records, we find that  $\text{Gain}(\text{Married})$  is 0,  $\text{Gain}(\text{Acct\_balance})$  is 1, and  $\text{Gain}(\text{Age})$  is 1.

**Figure 28.7**

Decision tree based on sample training data where the leaf nodes are represented by a set of RIDs of the partitioned records.

We can choose either Age or Acct\_balance since they both have the largest gain. Let us choose Age as the partitioning attribute. We add a node with label Age that has two branches, less than 25, and greater or equal to 25. Each branch partitions the remaining sample data such that one sample record belongs to each branch and hence one class. Two leaf nodes are created and we are finished. The final decision tree is pictured in Figure 28.7.

## 28.4 Clustering

The previous data mining task of classification deals with partitioning data based on using a preclassified training sample. However, it is often useful to partition data without having a training sample; this is also known as **unsupervised learning**. For example, in business, it may be important to determine groups of customers who have similar buying patterns, or in medicine, it may be important to determine groups of patients who show similar reactions to prescribed drugs. The goal of clustering is to place records into groups, such that records in a group are similar to each other and dissimilar to records in other groups. The groups are usually *disjoint*.

An important facet of clustering is the similarity function that is used. When the data is numeric, a similarity function based on distance is typically used. For example, the Euclidean distance can be used to measure similarity. Consider two  $n$ -dimensional data points (records)  $r_j$  and  $r_k$ . We can consider the value for the  $i$ th dimension as  $r_{ji}$  and  $r_{ki}$  for the two records. The Euclidean distance between points  $r_j$  and  $r_k$  in  $n$ -dimensional space is calculated as:

$$\text{Distance}(r_j, r_k) = \sqrt{|r_{j1} - r_{k1}|^2 + |r_{j2} - r_{k2}|^2 + \dots + |r_{jn} - r_{kn}|^2}$$

The smaller the distance between two points, the greater is the similarity as we think of them. A classic clustering algorithm is the  $k$ -means algorithm, Algorithm 28.4.

**Algorithm 28.4.** *k*-Means Clustering Algorithm**Input:** a database  $D$ , of  $m$  records,  $r_1, \dots, r_m$  and a desired number of clusters  $k$ **Output:** set of  $k$  clusters that minimizes the squared error criterion**Begin**

randomly choose  $k$  records as the centroids for the  $k$  clusters;  
 repeat  
   assign each record,  $r_i$ , to a cluster such that the distance between  $r_i$   
     and the cluster centroid (mean) is the smallest among the  $k$  clusters;  
   recalculate the centroid (mean) for each cluster based on the records  
     assigned to the cluster;  
 until no change;

**End;**

The algorithm begins by randomly choosing  $k$  records to represent the centroids (means),  $m_1, \dots, m_k$ , of the clusters,  $C_1, \dots, C_k$ . All the records are placed in a given cluster based on the distance between the record and the cluster mean. If the distance between  $m_i$  and record  $r_j$  is the smallest among all cluster means, then record  $r_j$  is placed in cluster  $C_i$ . Once all records have been initially placed in a cluster, the mean for each cluster is recomputed. Then the process repeats, by examining each record again and placing it in the cluster whose mean is closest. Several iterations may be needed, but the algorithm will converge, although it may terminate at a local optimum. The terminating condition is usually the squared-error criterion. For clusters  $C_1, \dots, C_k$  with means  $m_1, \dots, m_k$ , the error is defined as:

$$\text{Error} = \sum_{i=1}^k \sum_{\forall r_j \in C_i} \text{Distance}(r_j, m_i)^2$$

We will examine how Algorithm 28.4 works with the (two-dimensional) records in Figure 28.8. Assume that the number of desired clusters  $k$  is 2. Let the algorithm choose records with RID 3 for cluster  $C_1$  and RID 6 for cluster  $C_2$  as the initial cluster centroids. The remaining records will be assigned to one of those clusters during the first iteration of the repeat loop. The record with RID 1 has a distance from  $C_1$  of 22.4 and a distance from  $C_2$  of 32.0, so it joins cluster  $C_1$ . The record with RID 2 has

RID	Age	Years_of_service
1	30	5
2	50	25
3	50	15
4	25	5
5	30	10
6	55	25

**Figure 28.8**

Sample two-dimensional records for clustering example (the RID column is not considered).

a distance from  $C_1$  of 10.0 and a distance from  $C_2$  of 5.0, so it joins cluster  $C_2$ . The record with RID 4 has a distance from  $C_1$  of 25.5 and a distance from  $C_2$  of 36.6, so it joins cluster  $C_1$ . The record with RID 5 has a distance from  $C_1$  of 20.6 and a distance from  $C_2$  of 29.2, so it joins cluster  $C_1$ . Now, the new means (centroids) for the two clusters are computed. The mean for a cluster,  $C_i$ , with  $n$  records of  $m$  dimensions is the vector:

$$\bar{C}_i = \left( \frac{1}{n} \sum_{\forall r_j \in C_i} r_{ji}, \dots, \frac{1}{n} \sum_{\forall r_j \in C_i} r_{jm} \right)$$

The new mean for  $C_1$  is (33.75, 8.75) and the new mean for  $C_2$  is (52.5, 25). A second iteration proceeds and the six records are placed into the two clusters as follows: records with RIDs 1, 4, 5 are placed in  $C_1$  and records with RIDs 2, 3, 6 are placed in  $C_2$ . The mean for  $C_1$  and  $C_2$  is recomputed as (28.3, 6.7) and (51.7, 21.7), respectively. In the next iteration, all records stay in their previous clusters and the algorithm terminates.

Traditionally, clustering algorithms assume that the entire data set fits in main memory. More recently, researchers have developed algorithms that are efficient and are scalable for very large databases. One such algorithm is called BIRCH. BIRCH is a hybrid approach that uses both a hierarchical clustering approach, which builds a tree representation of the data, as well as additional clustering methods, which are applied to the leaf nodes of the tree. Two input parameters are used by the BIRCH algorithm. One specifies the amount of available main memory and the other is an initial threshold for the radius of any cluster. Main memory is used to store descriptive cluster information such as the center (mean) of a cluster and the radius of the cluster (clusters are assumed to be spherical in shape). The radius threshold affects the number of clusters that are produced. For example, if the radius threshold value is large, then few clusters of many records will be formed. The algorithm tries to maintain the number of clusters such that their radius is below the radius threshold. If available memory is insufficient, then the radius threshold is increased.

The BIRCH algorithm reads the data records sequentially and inserts them into an in-memory tree structure, which tries to preserve the clustering structure of the data. The records are inserted into the appropriate leaf nodes (potential clusters) based on the distance between the record and the cluster center. The leaf node where the insertion happens may have to split, depending upon the updated center and radius of the cluster and the radius threshold parameter. Additionally, when splitting, extra cluster information is stored, and if memory becomes insufficient, then the radius threshold will be increased. Increasing the radius threshold may actually produce a side effect of reducing the number of clusters since some nodes may be merged.

Overall, BIRCH is an efficient clustering method with a linear computational complexity in terms of the number of records to be clustered.

## 28.5 Approaches to Other Data Mining Problems

### 28.5.1 Discovery of Sequential Patterns

The discovery of sequential patterns is based on the concept of a sequence of itemsets. We assume that transactions such as the supermarket-basket transactions we discussed previously are ordered by time of purchase. That ordering yields a sequence of itemsets. For example, {milk, bread, juice}, {bread, eggs}, {cookies, milk, coffee} may be such a **sequence of itemsets** based on three visits by the same customer to the store. The **support** for a sequence  $S$  of itemsets is the percentage of the given set  $U$  of sequences of which  $S$  is a subsequence. In this example, {milk, bread, juice} {bread, eggs} and {bread, eggs} {cookies, milk, coffee} are considered **subsequences**. The problem of identifying sequential patterns, then, is to find all subsequences from the given sets of sequences that have a user-defined minimum support. The sequence  $S_1, S_2, S_3, \dots$  is a **predictor** of the fact that a customer who buys itemset  $S_1$  is likely to buy itemset  $S_2$  and then  $S_3$ , and so on. This prediction is based on the frequency (support) of this sequence in the past. Various algorithms have been investigated for sequence detection.

### 28.5.2 Discovery of Patterns in Time Series

**Time series** are sequences of events; each event may be a given fixed type of a transaction. For example, the closing price of a stock or a fund is an event that occurs every weekday for each stock and fund. The sequence of these values per stock or fund constitutes a time series. For a time series, one may look for a variety of patterns by analyzing sequences and subsequences as we did above. For example, we might find the period during which the stock rose or held steady for  $n$  days, or we might find the longest period over which the stock had a fluctuation of no more than 1% over the previous closing price, or we might find the quarter during which the stock had the most percentage gain or percentage loss. Time series may be compared by establishing measures of similarity to identify companies whose stocks behave in a similar fashion. Analysis and mining of time series is an extended functionality of temporal data management (see Chapter 26).

### 28.5.3 Regression

*Regression* is a special application of the classification rule. If a classification rule is regarded as a function over the variables that maps these variables into a target class variable, the rule is called a **regression rule**. A general application of regression occurs when, instead of mapping a tuple of data from a relation to a specific class, the value of a variable is predicted based on that tuple. For example, consider a relation

LAB\_TESTS (patient ID, test 1, test 2,  $\dots$ , test  $n$ )

which contains values that are results from a series of  $n$  tests for one patient. The target variable that we wish to predict is  $P$ , the probability of survival of the patient. Then the rule for regression takes the form:

$$\begin{aligned} &(\text{test 1 in range}_1) \text{ and } (\text{test 2 in range}_2) \text{ and } \dots (\text{test } n \text{ in range}_n) \Rightarrow P = x, \\ &\text{or } x < P \leq y \end{aligned}$$

The choice depends on whether we can predict a unique value of  $P$  or a range of values for  $P$ . If we regard  $P$  as a function:

$$P = f(\text{test 1, test 2, } \dots, \text{test } n)$$

the function is called a **regression function** to predict  $P$ . In general, if the function appears as

$$Y = f(X_1, X_2, \dots, X_n),$$

and  $f$  is linear in the domain variables  $x_i$ , the process of deriving  $f$  from a given set of tuples for  $\langle X_1, X_2, \dots, X_n, y \rangle$  is called **linear regression**. Linear regression is a commonly used statistical technique for fitting a set of observations or points in  $n$  dimensions with the target variable  $y$ .

Regression analysis is a very common tool for analysis of data in many research domains. The discovery of the function to predict the target variable is equivalent to a data mining operation.

### 28.5.4 Neural Networks

A **neural network** is a technique derived from artificial intelligence research that uses generalized regression and provides an iterative method to carry it out. Neural networks use the curve-fitting approach to infer a function from a set of samples. This technique provides a *learning approach*; it is driven by a test sample that is used for the initial inference and learning. With this kind of learning method, responses to new inputs may be able to be interpolated from the known samples. This interpolation, however, depends on the world model (internal representation of the problem domain) developed by the learning method.

Neural networks can be broadly classified into two categories: supervised and unsupervised networks. Adaptive methods that attempt to reduce the output error are **supervised learning** methods, whereas those that develop internal representations without sample outputs are called **unsupervised learning** methods.

Neural networks self-adapt; that is, they learn from information about a specific problem. They perform well on classification tasks and are therefore useful in data mining. Yet they are not without problems. Although they learn, they do not provide a good representation of *what* they have learned. Their outputs are highly quantitative and not easy to understand. As another limitation, the internal representations developed by neural networks are not unique. Also, in general, neural networks have trouble modeling time series data. Despite these shortcomings, they are popular and frequently used by several commercial vendors.

### 28.5.5 Genetic Algorithms

**Genetic algorithms** (GAs) are a class of randomized search procedures capable of adaptive and robust search over a wide range of search space topologies. Modeled after the adaptive emergence of biological species from evolutionary mechanisms, and introduced by Holland,<sup>6</sup> GAs have been successfully applied in such diverse fields as image analysis, scheduling, and engineering design.

Genetic algorithms extend the idea from human genetics of the four-letter alphabet (based on the A, C, T, G nucleotides) of the human DNA code. The construction of a genetic algorithm involves devising an alphabet that encodes the solutions to the decision problem in terms of strings of that alphabet. Strings are equivalent to individuals. A fitness function defines which solutions can survive and which cannot. The ways in which solutions can be combined are patterned after the cross-over operation of cutting and combining strings from a father and a mother. An initial population of a well-varied population is provided, and a game of evolution is played in which mutations occur among strings. They combine to produce a new generation of individuals; the fittest individuals survive and mutate until a family of successful solutions develops.

The solutions produced by GAs are distinguished from most other search techniques by the following characteristics:

- A GA search uses a set of solutions during each generation rather than a single solution.
- The search in the string-space represents a much larger parallel search in the space of encoded solutions.
- The memory of the search done is represented solely by the set of solutions available for a generation.
- A genetic algorithm is a randomized algorithm since search mechanisms use probabilistic operators.
- While progressing from one generation to the next, a GA finds near-optimal balance between knowledge acquisition and exploitation by manipulating encoded solutions.

Genetic algorithms are used for problem solving and clustering problems. Their ability to solve problems in parallel provides a powerful tool for data mining. The drawbacks of GAs include the large overproduction of individual solutions, the random character of the searching process, and the high demand on computer processing. In general, substantial computing power is required to achieve anything of significance with genetic algorithms.

---

<sup>6</sup>Holland's seminal work (1975) entitled *Adaptation in Natural and Artificial Systems* introduced the idea of genetic algorithms.



## 28.6 Applications of Data Mining

Data mining technologies can be applied to a large variety of decision-making contexts in business. In particular, areas of significant payoffs are expected to include the following:

- **Marketing.** Applications include analysis of consumer behavior based on buying patterns; determination of marketing strategies, including advertising, store location, and targeted mailing; segmentation of customers, stores, or products; and design of catalogs, store layouts, and advertising campaigns.
- **Finance.** Applications include analysis of creditworthiness of clients; segmentation of account receivables; performance analysis of finance investments like stocks, bonds, and mutual funds; evaluation of financing options; and fraud detection.
- **Manufacturing.** Applications involve optimization of resources like machines, personnel, and materials; and optimal design of manufacturing processes, shop-floor layouts, and product design, such as for automobiles based on customer requirements.
- **Healthcare.** Applications include discovery of patterns in radiological images, analysis of microarray (gene-chip) experimental data to cluster genes and to relate to symptoms or diseases, analysis of side effects of drugs and effectiveness of certain treatments, optimization of processes within a hospital, and analysis of the relationship between patient wellness data and doctor qualifications.

## 28.7 Commercial Data Mining Tools

Currently, commercial data mining tools use several common techniques to extract knowledge. These include association rules, clustering, neural networks, sequencing, and statistical analysis. We discussed these earlier. Also used are decision trees, which are a representation of the rules used in classification or clustering, and statistical analyses, which may include regression and many other techniques. Other commercial products use advanced techniques such as genetic algorithms, case-based reasoning, Bayesian networks, nonlinear regression, combinatorial optimization, pattern matching, and fuzzy logic. In this chapter, we have already discussed some of these.

Most data mining tools use the ODBC (Open Database Connectivity) interface. ODBC is an industry standard that works with databases; it enables access to data in most of the popular database programs such as Access, dBASE, Informix, Oracle, and SQL Server. Some of these software packages provide interfaces to specific database programs; the most common are Oracle, Access, and SQL Server. Most of the tools work in the Microsoft Windows environment and a few work in the UNIX operating system. The trend is for all products to operate under the Microsoft

Windows environment. One tool, Data Surveyor, mentions ODMG compliance; see Chapter 12, where we discussed the ODMG object-oriented standard.

In general, these programs perform sequential processing in a single machine. Many of these products work in the client/server mode. Some products incorporate parallel processing in parallel computer architectures and work as a part of online analytical processing (OLAP) tools.

### 28.7.1 User Interface

Most of the tools run in a graphical user interface (GUI) environment. Some products include sophisticated visualization techniques to view data and rules (for example, SGI's MineSet), and are even able to manipulate data this way interactively. Text interfaces are rare and are more common in tools available for UNIX, such as IBM's Intelligent Miner.

### 28.7.2 Application Programming Interface

Usually, the application programming interface (API) is an optional tool. Most products do not permit using their internal functions. However, some of them allow the application programmer to reuse their code. The most common interfaces are C libraries and dynamic link libraries (DLLs). Some tools include proprietary database command languages.

Table 28.1 lists 11 representative data mining tools. To date, there are hundreds of commercial data mining products available worldwide. Non-U.S. products include Data Surveyor from the Netherlands and PolyAnalyst from Russia.

### 28.7.3 Future Directions

Data mining tools are continually evolving, building on ideas from the latest scientific research. Many of these tools incorporate the latest algorithms taken from artificial intelligence (AI), statistics, and optimization. Currently, fast processing is done using modern database techniques—such as distributed processing—in client/server architectures, in parallel databases, and in data warehousing. For the future, the trend is toward developing Internet capabilities more fully. Additionally, hybrid approaches will become commonplace, and processing will be done using all resources available. Processing will take advantage of both parallel and distributed computing environments. This shift is especially important because modern databases contain very large amounts of information.

The primary direction for data mining is to analyze terabytes and petabytes of data in the so-called big data systems that we presented in Chapter 25. These systems are being equipped with their own tools and libraries for data mining, such as Mahout, which runs on top of Hadoop, which we described in detail. The data mining area will also be closely tied to data that will be housed in the cloud in data warehouses

**Table 28.1** Some Representative Data Mining Tools

Company	Product	Technique	Platform	Interface*
AcknoSoft	Kate	Decision trees, case-based reasoning	Windows UNIX	Microsoft Access
Angoss	Knowledge SEEKER	Decision trees, statistics	Windows	ODBC
Business Objects	Business Miner	Neural nets, machine learning	Windows	ODBC
CrossZ	QueryObject	Statistical analysis, optimization algorithm	Windows MVS UNIX	ODBC
Data Distilleries	Data Surveyor	Comprehensive; can mix different types of data mining	UNIX	ODBC and ODMG-compliant
DBMiner Technology Inc.	DBMiner	OLAP analysis, associations, classification, clustering algorithms	Windows	Microsoft 7.0 OLAP
IBM	Intelligent Miner	Classification, association rules, predictive models	UNIX (AIX)	IBM DB2
Megaputer Intelligence	PolyAnalyst	Symbolic knowledge acquisition, evolutionary programming	Windows OS/2	ODBC Oracle DB2
NCR	Management Discovery Tool (MDT)	Association rules	Windows	ODBC
Purple Insight	MineSet	Decision trees, association rules	UNIX (Irix)	Oracle Sybase Informix
SAS	Enterprise Miner	Decision trees, association rules, Nneural nets, regression, clustering	UNIX (Solaris) Windows Macintosh	ODBC Oracle AS/400

\*ODBC: Open Database Connectivity

ODMG: Object Data Management Group

and brought into service for mining operations as needed using OLAP (online analytical processing) servers. Not only are multimedia databases growing, but, in addition, image storage and retrieval are slow operations. Also, the cost of secondary storage is decreasing, so massive information storage will be feasible, even for small companies. Thus, data mining programs will have to deal with larger sets of data of more companies.

Most of data mining software will use the ODBC standard to extract data from business databases; proprietary input formats can be expected to disappear. There is a definite need to include nonstandard data, including images and other multimedia data, as source data for data mining.

## 28.8 Summary

In this chapter, we surveyed the important discipline of data mining, which uses database technology to discover additional knowledge or patterns in the data. We gave an illustrative example of knowledge discovery in databases, which has a wider scope than data mining. For data mining, among the various techniques, we focused on the details of association rule mining, classification, and clustering. We presented algorithms in each of these areas and illustrated with examples of how those algorithms work.

A variety of other techniques, including the AI-based neural networks and genetic algorithms, were also briefly discussed. Active research is ongoing in data mining, and we have outlined some of the expected research directions. In the future database technology products market, a great deal of data mining activity is expected. We summarized 11 out of several hundred data mining tools available; future research is expected to extend the number and functionality significantly.

## Review Questions

- 28.1. What are the different phases of the knowledge discovery from databases? Describe a complete application scenario in which new knowledge may be mined from an existing database of transactions.
- 28.2. What are the goals or tasks that data mining attempts to facilitate?
- 28.3. What are the five types of knowledge produced from data mining?
- 28.4. What are association rules as a type of knowledge? Define *support* and *confidence*, and use these definitions to define an association rule.
- 28.5. What is the downward closure property? How does it aid in developing an efficient algorithm for finding association rules; that is, with regard to finding large itemsets?
- 28.6. What was the motivating factor for the development of the FP-tree algorithm for association rule mining?
- 28.7. Describe an association rule among hierarchies and provide an example.
- 28.8. What is a negative association rule in the context of the hierarchy in Figure 28.3?
- 28.9. What are the difficulties of mining association rules from large databases?
- 28.10. What are classification rules, and how are decision trees related to them?
- 28.11. What is entropy, and how is it used in building decision trees?
- 28.12. How does clustering differ from classification?
- 28.13. Describe neural networks and genetic algorithms as techniques for data mining. What are the main difficulties in using these techniques?

## Exercises

**28.14.** Apply the apriori algorithm to the following data set.

Trans_id	Items_purchased
101	milk, bread, eggs
102	milk, juice
103	juice, butter
104	milk, bread, eggs
105	coffee, eggs
106	coffee
107	coffee, juice
108	milk, bread, cookies, eggs
109	cookies, butter
110	milk, bread

The set of items is {milk, bread, cookies, eggs, butter, coffee, juice}. Use 0.2 for the minimum support value.

- 28.15.** Show two rules that have a confidence of 0.7 or greater for an itemset containing three items from Exercise 28.14.
- 28.16.** For the partition algorithm, prove that any frequent itemset in the database must appear as a local frequent itemset in at least one partition.
- 28.17.** Show the FP-tree that would be made for the data from Exercise 28.14.
- 28.18.** Apply the FP-growth algorithm to the FP-tree from Exercise 28.17 and show the frequent itemsets.
- 28.19.** Apply the classification algorithm to the following set of data records. The class attribute is Repeat\_customer.

RID	Age	City	Gender	Education	Repeat_customer
101	20 ... 30	NY	F	college	YES
102	20 ... 30	SF	M	graduate	YES
103	31 ... 40	NY	F	college	YES
104	51 ... 60	NY	F	college	NO
105	31 ... 40	LA	M	high school	NO
106	41 ... 50	NY	F	college	YES
107	41 ... 50	NY	F	graduate	YES
108	20 ... 30	LA	M	college	YES
109	20 ... 30	NY	F	high school	NO
110	20 ... 30	NY	F	college	YES

**28.20.** Consider the following set of two-dimensional records:

RID	Dimension1	Dimension2
1	8	4
2	5	4
3	2	4
4	2	6
5	2	8
6	8	6

Also consider two different clustering schemes: (1) where Cluster<sub>1</sub> contains records {1, 2, 3} and Cluster<sub>2</sub> contains records {4, 5, 6}, and (2) where Cluster<sub>1</sub> contains records {1, 6} and Cluster<sub>2</sub> contains records {2, 3, 4, 5}. Which scheme is better and why?

**28.21.** Use the  $k$ -means algorithm to cluster the data from Exercise 28.20. We can use a value of 3 for  $K$ , and we can assume that the records with RIDs 1, 3, and 5 are used for the initial cluster centroids (means).

**28.22.** The  $k$ -means algorithm uses a similarity metric of distance between a record and a cluster centroid. If the attributes of the records are not quantitative but categorical in nature, such as Income\_level with values {low, medium, high} or Married with values {Yes, No} or State\_of\_residence with values {Alabama, Alaska, ... , Wyoming}, then the distance metric is not meaningful. Define a more suitable similarity metric that can be used for clustering data records that contain categorical data.

## Selected Bibliography

Literature on data mining comes from several fields, including statistics, mathematical optimization, machine learning, and artificial intelligence. Chen et al. (1996) give a good summary of the database perspective on data mining. The book by Han and Kamber (2006) is an excellent text and describes in detail the different algorithms and techniques used in the data mining area. Work at IBM Almaden Research has produced a large number of early concepts and algorithms as well as results from some performance studies. Agrawal et al. (1993) report the first major study on association rules. Their apriori algorithm for market-basket data in Agrawal and Srikant (1994) is improved by using partitioning in Savasere et al. (1995); Toivonen (1996) proposes sampling as a way to reduce the processing effort. Cheung et al. (1996) extends the partitioning to distributed environments; Lin and Dunham (1998) propose techniques to overcome problems with data skew. Agrawal et al. (1993b) discuss the performance perspective on association rules. Mannila et al. (1994), Park et al. (1995), and Amir et al. (1997) present additional efficient algorithms related to association rules. Han et al. (2000) present the FP-tree algorithm

discussed in this chapter. Srikant and Agrawal (1995) proposes mining generalized rules. Savasere et al. (1998) present the first approach to mining negative associations. Agrawal et al. (1996) describe the Quest system at IBM. Sarawagi et al. (1998) describe an implementation where association rules are integrated with a relational database management system. Piatetsky-Shapiro and Frawley (1992) have contributed papers from a wide range of topics related to knowledge discovery. Zhang et al. (1996) present the BIRCH algorithm for clustering large databases. Information about decision tree learning and the classification algorithm presented in this chapter can be found in Mitchell (1997).

Adriaans and Zantinge (1996), Fayyad et al. (1997), and Weiss and Indurkha (1998) are books devoted to the different aspects of data mining and its use in prediction. The idea of genetic algorithms was proposed by Holland (1975); a good survey of genetic algorithms appears in Srinivas and Patnaik (1994). Neural networks have a vast literature; a comprehensive introduction is available in Lippman (1987).

Tan, Steinbach, and Kumar (2006) provides a comprehensive introduction to data mining and has a detailed set of references. Readers are also advised to consult proceedings of two prominent annual conferences in data mining: the Knowledge Discovery and Data Mining Conference (KDD), which has been running since 1995, and the SIAM International Conference on Data Mining (SDM), which has been running since 2001. Links to past conferences may be found at <http://dblp.uni-trier.de>.



## Overview of Data Warehousing and OLAP

Data warehouses are databases that store and maintain analytical data separately from transaction-oriented databases for the purpose of decision support. Regular transaction-oriented databases store data for a limited period of time before the data loses its immediate usefulness and it is archived. On the other hand, data warehouses tend to keep years' worth of data in order to enable analysis of historical data. They provide storage, functionality, and responsiveness to queries beyond the capabilities of transaction-oriented databases. Accompanying this ever-increasing power is a great demand to improve the data access performance of databases. In modern organizations, users of data are often completely removed from the data sources. Many people only need read-access to data, but still need fast access to a larger volume of data than can conveniently be downloaded to their desktops. Often such data comes from multiple databases. Because many of the analyses performed are recurrent and predictable, software vendors and systems support staff are designing systems to support these functions. Data warehouses are modeled and structured differently, they use different types of technologies for storage and retrieval, and they are used by different types of users than transaction-oriented databases. Presently there is a great need to provide decision makers from middle management upward with information at the correct level of detail to support decision making. *Data warehousing*, *online analytical processing* (OLAP), and *data mining* provide this functionality. We gave an introduction to data mining techniques in Chapter 28. In this chapter, we give a broad overview of data warehousing and OLAP technologies.

## 29.1 Introduction, Definitions, and Terminology

In Chapter 1, we defined a *database* as a collection of related data and a *database system* as a database and database software together. A data warehouse is also a collection of information as well as a supporting system. However, a clear distinction exists. Traditional databases are transactional (relational, object-oriented, network, or hierarchical). *Data warehouses* have the distinguishing characteristic that they are mainly intended for decision-support applications. They are optimized for data retrieval, not routine transaction processing.

Because data warehouses have been developed in numerous organizations to meet particular needs, there is no single, canonical definition of the term *data warehouse*. Professional magazine articles and books in the popular press have elaborated on the meaning in a variety of ways. Vendors have capitalized on the popularity of the term to help market a variety of related products, and consultants have provided a large variety of services, all under the data warehousing banner. However, data warehouses are distinct from traditional databases in their structure, functioning, performance, and purpose.

W. H. Inmon<sup>1</sup> characterized a **data warehouse** as *a subject-oriented, integrated, nonvolatile, time-variant collection of data in support of management's decisions*. Data warehouses provide access to data for complex analysis, knowledge discovery, and decision making through **ad hoc** and canned queries. **Canned queries** refer to *a-priori* defined queries with parameters that may recur with high frequency. They support high-performance demands on an organization's data and information. Several types of applications—OLAP, DSS, and data mining applications—are supported. We define each of these next.

**OLAP (online analytical processing)** is a term used to describe the analysis of complex data from the data warehouse. In the hands of skilled knowledge workers, OLAP tools enable quick and straightforward querying of the analytical data stored in data warehouses and **data marts** (analytical databases similar to data warehouses but with a defined narrow scope).

**DSS (decision-support systems)**, also known as **EIS (or MIS)—executive information systems (or management information systems)**, not to be confused with enterprise integration systems—support an organization's leading decision makers with higher-level (analytical) data for complex and important decisions. Data mining (which we discussed in Chapter 28) is used for *knowledge discovery*, the ad hoc process of searching data for unanticipated new knowledge (not unlike looking for pearls of wisdom in an ocean of data).

Traditional databases support **online transaction processing (OLTP)**, which includes insertions, updates, and deletions while also supporting information query requirements. Traditional relational databases are optimized to process queries that

---

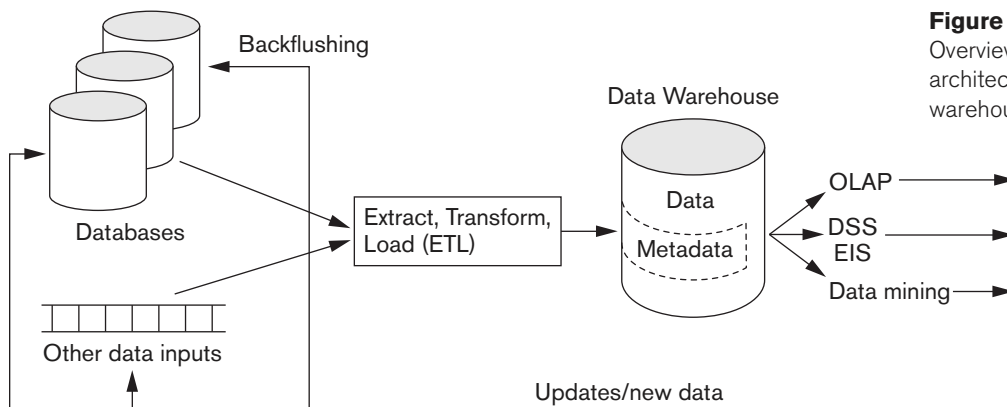
<sup>1</sup>Inmon (1992) is credited with initially using the term *warehouse*. Inmon et al. (2008) is titled "DW 2.0: The architecture for the next generation of Data Warehousing."

may touch a small part of the database and transactions that deal with insertions or updates of a few tuples per relation to process. Thus, they cannot be optimized for OLAP, DSS, or data mining. By contrast, data warehouses are designed precisely to support efficient extraction, processing, and presentation for analytic and decision-making purposes. In comparison to traditional databases, data warehouses generally contain very large amounts of data from multiple sources that may include databases from different data models and sometimes files acquired from independent systems and platforms.

## 29.2 Characteristics of Data Warehouses

To discuss data warehouses and distinguish them from transactional databases calls for an appropriate data model. The multidimensional data model (explained in more detail in Section 29.3) is a good fit for OLAP and decision-support technologies. In contrast to multidatabases, which provide access to disjoint and usually heterogeneous databases, a data warehouse is frequently a store of integrated data from multiple sources, processed for storage in a multidimensional model. Unlike most transactional databases, data warehouses typically support time series and trend analyses along with what-if or predictive-type analyses, all of which require more historical data than is generally maintained in transactional databases.

Compared with transactional databases, data warehouses are nonvolatile. This means that information in the data warehouse is typically not subject to modification and is often referred to as read/append/purge only. A data warehouse may be regarded as non-real-time with periodic insertions. In transactional systems, transactions are the unit and are the agent of change to the database; by contrast, data warehouse information is much more coarse-grained and is refreshed according to a careful choice of refresh policy, usually incremental. Warehouse insertions are handled by the warehouse's **ETL (extract, transform, load)** process, which does a large amount of preprocessing and which is shown in Figure 29.1. We can also describe



**Figure 29.1**  
Overview of the general architecture of a data warehouse.

data warehousing more generally as *a collection of decision-support technologies aimed at enabling the knowledge worker (executive, manager, analyst) to make better and faster decisions*.<sup>2</sup> Figure 29.1 gives an overview of the conceptual structure of a data warehouse. It shows the entire data warehousing process, which includes possible cleaning and reformatting of data before loading it into the warehouse. This process is handled by tools known as ETL (extraction, transformation, and loading) tools. At the back end of the process, OLAP, data mining, and DSS may generate new relevant information such as rules (or additional meta-data); this information is shown in Figure 29.1 as going back as additional data inputs into the warehouse. The figure also shows that data sources may include files.

The important characteristics of data warehouses that accompanied the definition of the term OLAP in 1993 included the following, and they are applicable even today:<sup>3</sup>

- Multidimensional conceptual view
- Unlimited dimensions and aggregation levels
- Unrestricted cross-dimensional operations
- Dynamic sparse matrix handling
- Client/server architecture
- Multiuser support
- Accessibility
- Transparency
- Intuitive data manipulation
- Inductive and deductive analysis
- Flexible distributed reporting

Because they encompass large volumes of data, data warehouses are generally an order of magnitude (sometimes two orders of magnitude) larger than the source databases. The sheer volume of data (likely to be in terabytes or even petabytes) is an issue that has been dealt with through enterprise-wide data warehouses, virtual data warehouses, logical data warehouses, and data marts:

- **Enterprise-wide data warehouses** are huge projects requiring massive investment of time and resources.
- **Virtual data warehouses** provide views of operational databases that are materialized for efficient access.
- **Logical data warehouses** use data federation, distribution, and virtualization techniques.
- **Data marts** generally are targeted to a subset of the organization, such as a department, and are more tightly focused.

---

<sup>2</sup>Chaudhuri and Dayal (1997) provide an excellent tutorial on the topic, with this as a starting definition.

<sup>3</sup>Codd and Salley (1993) coined the term *OLAP* and mentioned the characteristics listed here.

Other terms frequently encountered in the context of data warehousing are as follows:

- **Operational data store (ODS):** This term is commonly used for intermediate form of databases before they are cleansed, aggregated, and transformed into a data warehouse.
- **Analytical data store (ADS):** Those are the database that are built for the purpose of conducting data analysis. Typically, ODSs are reconfigured and repurposed into ADSs through the processes of cleansing, aggregation, and transformation.

### 29.3 Data Modeling for Data Warehouses

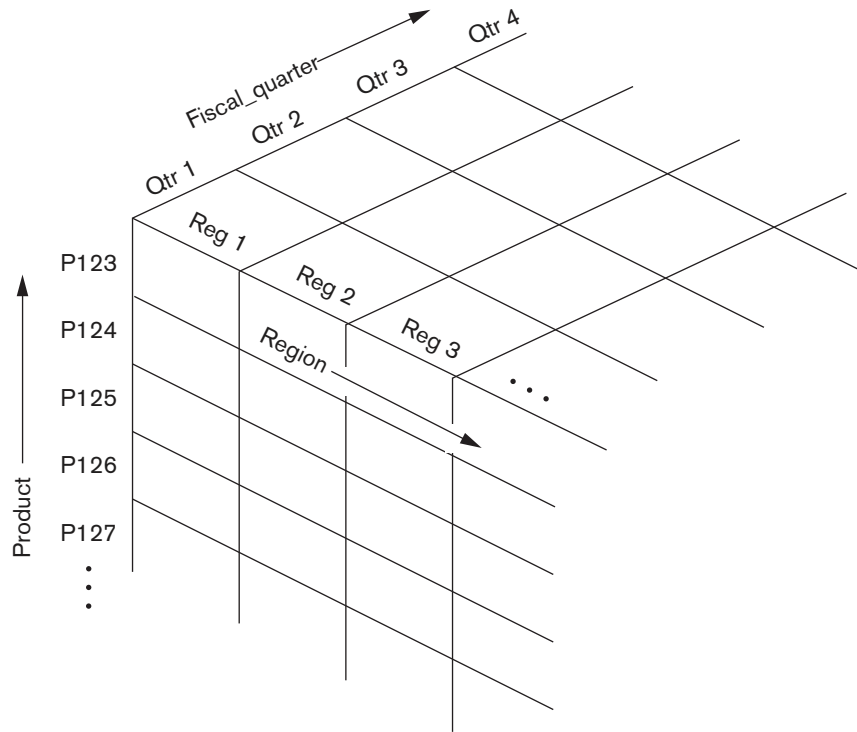
Multidimensional models take advantage of inherent relationships in data to populate data in multidimensional matrices called *data cubes*. (These may be called *hyper-cubes* if they have more than three dimensions.) For data that lends itself to multidimensional modeling, query performance in multidimensional matrices can be much better than in the relational data model. Three examples of dimensions in a corporate data warehouse are the corporation’s fiscal periods, products, and regions.

A standard spreadsheet is a two-dimensional matrix. One example would be a spreadsheet of regional sales by product for a particular time period. Products could be shown as rows, with columns comprising sales revenues for each region. (Figure 29.2 shows this two-dimensional organization.) Adding a time dimension, such as an organization’s fiscal quarters, would produce a three-dimensional matrix, which could be represented using a data cube.

Figure 29.3 shows a three-dimensional data cube that organizes product sales data by fiscal quarters and sales regions. Each cell could contain data for a specific product,

		Region			
		Reg 1	Reg 2	Reg 3	...
Product	P123				
	P124				
	P125				
	P126				
	...				

**Figure 29.2**  
A two-dimensional matrix model.

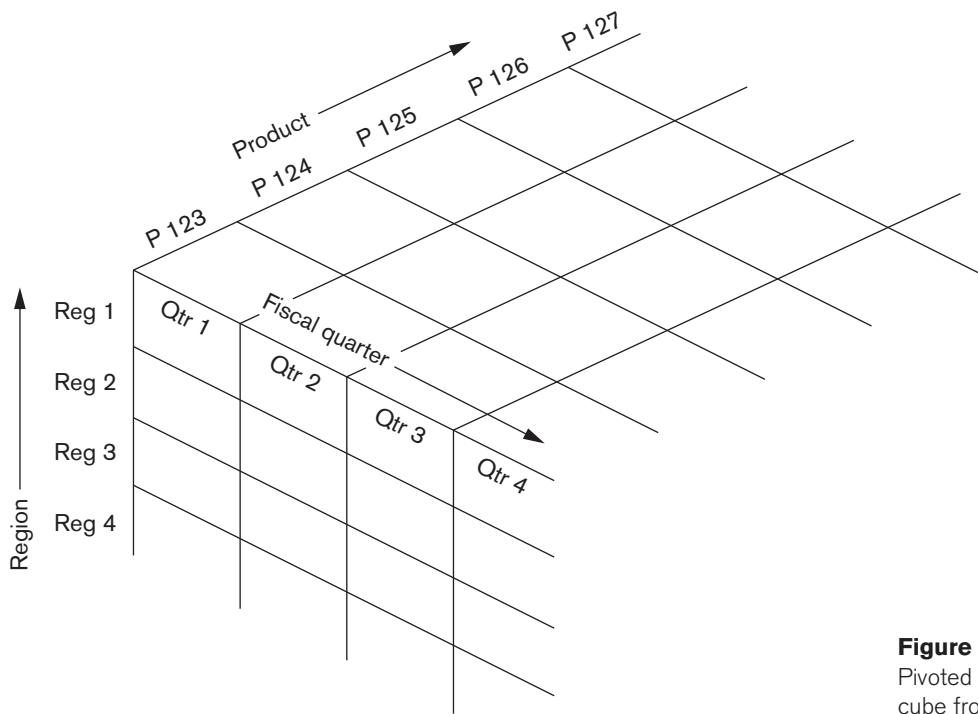
**Figure 29.3**

A three-dimensional data cube model.

specific fiscal quarter, and specific region. By including additional dimensions, a data hypercube could be produced, although more than three dimensions cannot be easily visualized or graphically presented. The data can be queried directly in any combination of dimensions, thus bypassing complex database queries. Tools exist for viewing data according to the user's choice of dimensions.

Changing from one-dimensional hierarchy (orientation) to another is easily accomplished in a data cube with a technique called **pivoting** (also called *rotation*). In this technique, the data cube can be thought of as rotating to show a different orientation of the axes. For example, you might pivot the data cube to show regional sales revenues as rows, the fiscal quarter revenue totals as columns, and the company's products in the third dimension (Figure 29.4). Hence, this technique is equivalent to having a regional sales table for each product separately, where each table shows quarterly sales for that product region by region. The term **slice** is used to refer to a two-dimensional view of a three- or higher-dimensional cube. The Product vs. Region 2-D view shown in Figure 29.2 is a slice of the 3-D cube shown in Figure 29.3. The popular term "slice and dice" implies a systematic reduction of a body of data into smaller chunks or views so that the information is made visible from multiple angles or viewpoints.

Multidimensional models lend themselves readily to hierarchical views in what is known as roll-up display and drill-down display. A **roll-up display** moves up the



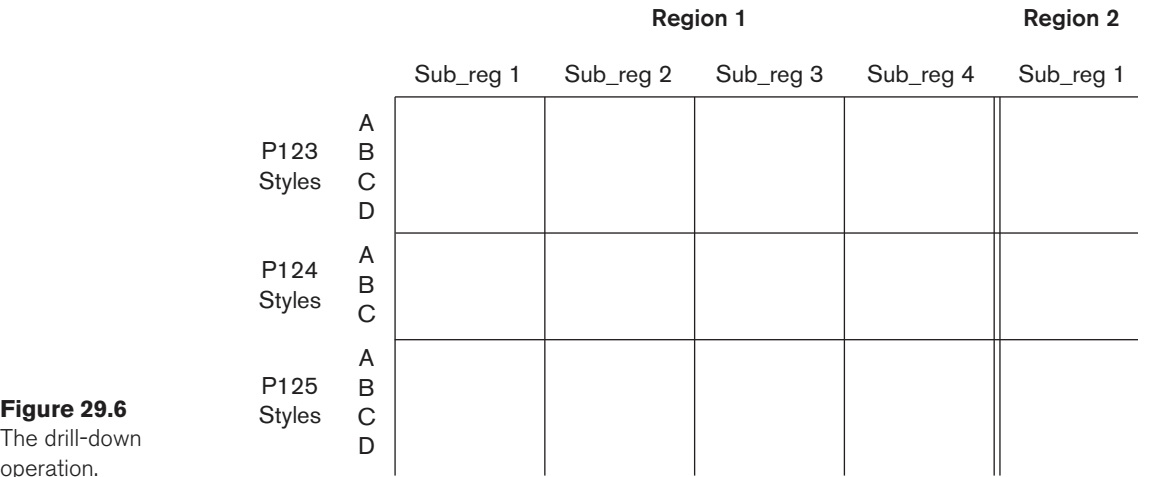
**Figure 29.4**  
Pivoted version of the data cube from Figure 29.3.

hierarchy, grouping into larger units along a dimension (for example, summing weekly data by quarter or by year). Figure 29.5 shows a roll-up display that moves from individual products to a coarser grain of product categories. Shown in Figure 29.6, a **drill-down display** provides the opposite capability, furnishing a finer-grained view, perhaps disaggregating country sales by region and then

		Region →		
		Region 1	Region 2	Region 3
Product categories ↓	Products 1XX			
	Products 2XX			
	Products 3XX			
	Products 4XX			

**Figure 29.5**  
The roll-up operation.



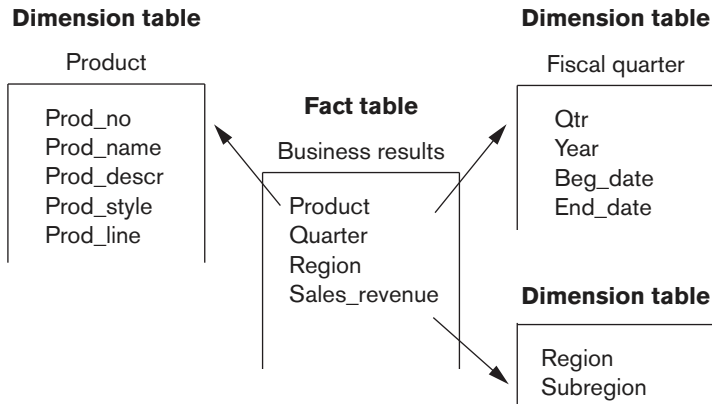


**Figure 29.6**  
The drill-down operation.

regional sales by subregion and also breaking up products by styles. Typically, in a warehouse, the **drill-down** capability is limited to the lowest level of aggregated data stored in the warehouse. For example, compared to the data shown in Figure 29.6, lower- level data will correspond to something like “the total sales for style P123 substyle A color Black in zipcode 30022 of sub-region 1.” That level of aggregation may have been kept in the ODS. Some DBMSs like Oracle offer the “nested table” concept, which enables access to lower levels of data and thus makes the drill-down penetrate deeper.

The **multidimensional model** (also called the **dimensional model**)-involves two types of tables: dimension tables and fact tables. A **dimension table** consists of tuples of attributes of the dimension. A **fact table** can be thought of as having tuples, one per a recorded fact. This fact contains some measured or observed variable(s) and identifies it (them) with pointers to dimension tables. The fact table contains the data, and the dimensions identify each tuple in that data. Another way to look at a fact table is as an agglomerated view of the transaction data whereas each dimension table represents so-called “master data” that those transactions belonged to. In multidimensional database systems, the multidimensional model has been implemented as specialized software system known as a *multidimensional database*, which we do not discuss. Our treatment of the multidimensional model is based on storing the warehouse as a relational database in an RDBMS.

Figure 29.7 shows an example of a fact table that can be viewed from the perspective of multi-dimension tables. Two common multidimensional schemas are the star schema and the snowflake schema. The **star schema** consists of a fact table with a single table for each dimension (Figure 29.7). The **snowflake schema** is a variation on the star schema in which the dimensional tables from a star schema are organized

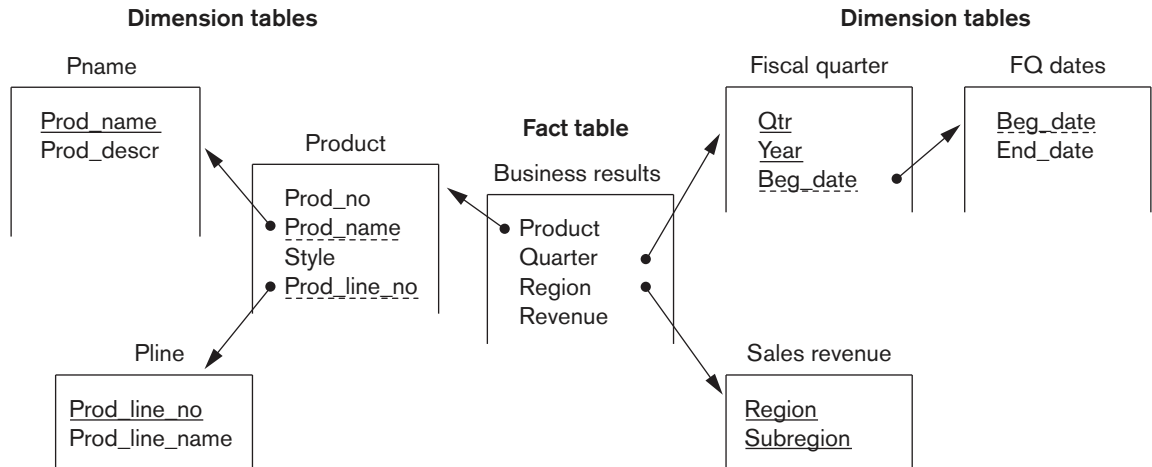


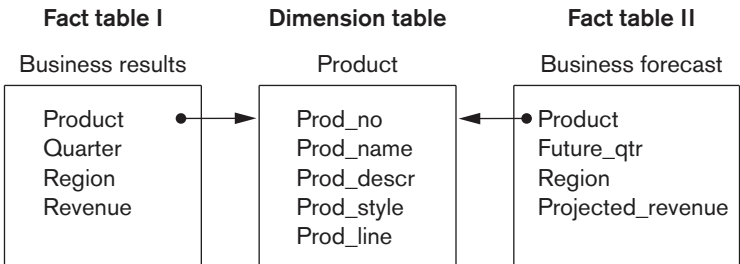
**Figure 29.7**  
A star schema with fact and dimensional tables.

into a hierarchy by normalizing them (Figure 29.8). A **fact constellation** is a set of fact tables that share some dimension tables. Figure 29.9 shows a fact constellation with two fact tables, business results and business forecast. These share the dimension table called product. Fact constellations limit the possible queries for the warehouse.

Data warehouse storage also utilizes indexing techniques to support high-performance access (see Chapter 17 for a discussion of indexing). A technique called **bitmap indexing** constructs a bit vector for each value in a domain (column) being indexed. It works very well for domains of low cardinality. There is a 1 bit placed in

**Figure 29.8**  
A snowflake schema.





**Figure 29.9**  
A fact constellation.

the  $j$ th position in the vector if the  $j$ th row contains the value being indexed. For example, imagine an inventory of 100,000 cars with a bitmap index on car size. If there are four car sizes—economy, compact, mid-size, and full-size—there will be four bit vectors, each containing 100,000 bits (12.5kbytes) for a total index size of 50K. Bitmap indexing can provide considerable input/output and storage space advantages in low-cardinality domains. With bit vectors, a bitmap index can provide dramatic improvements in comparison, aggregation, and join performance. We showed an example of a query on a star schema in Section 19.8, and we also showed the star schema’s transformation for efficient execution that uses bitmap indexes.

In a star schema, dimensional data can be indexed to tuples in the fact table by **join indexing**. Join indexes are traditional indexes used to maintain relationships between primary key and foreign key values. They relate the values of a dimension of a star schema to rows in the fact table. Consider a sales fact table that has city and fiscal quarter as dimensions. If there is a join index on city, for each city the join index maintains the tuple IDs of tuples containing that city. Join indexes may involve multiple dimensions.

Data warehouse storage can facilitate access to summary data by taking further advantage of the nonvolatility of data warehouses and a degree of predictability of the analyses that will be performed using them. Two approaches have been used: (1) smaller tables that include summary data such as quarterly sales or revenue by product line, and (2) encoding of level (for example, weekly, quarterly, annual) into existing tables. The overhead of creating and maintaining such aggregations would likely be excessive in a dynamically changing, transaction-oriented database.

The purpose of **master data management (MDM)**, a popular concept within enterprises, is to define the standards, processes, policies, and governance related to the critical data entities of the organization. The dimension tables—which in a data warehouse physicalize concepts, such as customers, regions and product categories—represent essentially the master data. Since dimensions are shared across multiple facts or reporting data marts, data warehouse designers typically must spend a considerable amount of time cleansing and harmonizing these dimensions (i.e., reconciling definitional and notional differences across multiple source systems that the dimension data comes from). As such, table structures containing these dimensions become good candidates for special copies of master data that can be used in other environments.

## 29.4 Building a Data Warehouse

In constructing a data warehouse, builders should take a broad view of the anticipated use of the warehouse. There is no way to anticipate all possible queries or analyses during the design phase. However, the design should specifically support **ad hoc querying**; that is, accessing data with any combination of values for the attributes that would be meaningful in the dimension or fact tables. For example, a marketing-intensive consumer-products company would require different ways of organizing the data warehouse than would a nonprofit charity focused on fund raising. An appropriate schema should be chosen that reflects anticipated usage.

Acquisition of data for the warehouse involves the following steps:

1. The data must be extracted from multiple, heterogeneous sources; for example, databases or other data feeds such as those containing financial market data or environmental data.
2. Data must be formatted for consistency within the warehouse. Names, meanings, and domains of data from unrelated sources must be reconciled. For instance, subsidiary companies of a large corporation may have different fiscal calendars with quarters ending on different dates, making it difficult to aggregate financial data by quarter. Various credit cards may report their transactions differently, making it difficult to compute all credit sales. These format inconsistencies must be resolved.
3. The data must be cleaned to ensure validity. Data cleaning is an involved and complex process that has been identified as the largest labor-demanding component of data warehouse construction. For input data, cleaning must occur before the data is loaded into the warehouse. Since input data must be examined and formatted consistently, data warehouse builders should take this opportunity to check each input for validity and quality. Recognizing erroneous and incomplete data is difficult to automate, and cleaning that requires automatic error correction can be even tougher. Some aspects, such as domain checking, are easily coded into data cleaning routines, but automatic recognition of other data problems can be more challenging. (For example, one might require that City = 'San Francisco' together with State = 'CT' be recognized as an incorrect combination.) After such problems have been taken care of, similar data from different sources must be coordinated for loading into the warehouse. As data managers in the organization discover that their data is being cleaned for input into the warehouse, they will likely want to upgrade their data with the cleaned data. The process of returning cleaned data to the source is called **backflushing** (see Figure 29.1).
4. The data must be fitted into the data model of the warehouse. Data from the various sources must be represented in the data model of the warehouse. Data may have to be converted from relational, object-oriented, or legacy databases (network and/or hierarchical) to a multidimensional model.
5. The data must be loaded into the warehouse. The sheer volume of data in the warehouse makes loading the data a significant task. Monitoring tools

for loads as well as methods to recover from incomplete or incorrect loads are required. With the huge volume of data in the warehouse, incremental updating is usually the only feasible approach. The refresh policy will probably emerge as a compromise that takes into account the answers to the following questions:

- ❑ How up-to-date must the data be?
- ❑ Can the warehouse go offline, and for how long?
- ❑ What are the data interdependencies?
- ❑ What is the storage availability?
- ❑ What are the distribution requirements (such as for replication and partitioning)?
- ❑ What is the loading time (including cleaning, formatting, copying, transmitting, and overhead such as index rebuilding)?

Data in a warehouse can come from multiple sources, geographies, and/or time zones. Data loads, therefore, need to be carefully planned and staged. The order in which data is loaded into the warehouse is critical; failure to load data in the correct order could lead to integrity constraints or semantic rule violations, both of which could cause load failures. For example, master data (whether new or changed) such as Customer and Product must be loaded prior to the transactions that contain them; and invoice data must be loaded before the billing data that references it.

As we have said, databases must strike a balance between efficiency in transaction processing and support for query requirements (ad hoc user requests), but a data warehouse is typically optimized for access from a decision maker's needs. Data storage in a data warehouse reflects this specialization and involves the following processes:

- Storing the data according to the data model of the warehouse
- Creating and maintaining required data structures
- Creating and maintaining appropriate access paths
- Providing for time-variant data as new data are added
- Supporting the updating of warehouse data
- Refreshing the data
- Purging data

Although adequate time can be devoted initially to constructing the warehouse, the sheer volume of data in the warehouse generally makes it impossible to simply reload the warehouse in its entirety later on. Alternatives include selective (partial) refreshing of data and separate warehouse versions (which requires double storage capacity for the warehouse). When the warehouse uses an incremental data refreshing mechanism, data may need to be purged periodically; for example, a warehouse that maintains data on the previous twelve business quarters may periodically purge its data each year, or even each quarter.

Data warehouses must also be designed with full consideration of the environment in which they will reside. Important design considerations include the following:

- Usage projections
- The fit of the data model
- Characteristics of available sources
- Design of the meta-data component
- Modular component design
- Design for manageability and change
- Considerations of distributed and parallel architecture

We discuss each of these in turn. Warehouse design is initially driven by usage projections; that is, by expectations about who will use the warehouse and how they will use it. Choice of a data model to support this usage is a key initial decision. Usage projections and the characteristics of the warehouse's data sources are both taken into account. Modular design is a practical necessity to allow the warehouse to evolve with the organization and its information environment. Additionally, a well-built data warehouse must be designed for maintainability, enabling the warehouse managers to plan for and manage change effectively while providing optimal support to users.

You may recall the term *meta-data* from Chapter 1; meta-data was defined as the description of a database; this description includes the database's schema definition. The **meta-data repository** is a key data warehouse component. The meta-data repository includes both technical and business meta-data. The first, **technical meta-data**, covers details of acquisition, processing, storage structures, data descriptions, warehouse operations and maintenance, and access support functionality. The second, **business meta-data**, includes the relevant business rules and organizational details supporting the warehouse.

The architecture of the organization's distributed computing environment is a major determining characteristic for the design of the warehouse. There are two basic distributed architectures: the distributed warehouse and the federated warehouse. For a **distributed warehouse**, all the issues of distributed databases are relevant; for example, replication, partitioning, communications, and consistency concerns. A distributed architecture can provide benefits particularly important to warehouse performance, such as improved load balancing, scalability of performance, and higher availability. A single replicated meta-data repository would reside at each distribution site. The idea of the **federated warehouse** is like that of the federated database: a decentralized confederation of autonomous data warehouses, each with its own meta-data repository. Given the magnitude of the challenge inherent to data warehouses, it is likely that such federations will consist of smaller scale components, such as data marts.

Businesses are becoming dissatisfied with the traditional data warehousing techniques and technologies. New analytic requirements are driving new analytic appliances; examples include Netezza of IBM, Greenplum of EMC, Hana of SAP, and

ParAccel of Tableau Software. Big data analytics have driven Hadoop and other specialized databases such as graph and key-value stores into the next generation of data warehousing (see Chapter 25 for a discussion of big data technology based on Hadoop). Data virtualization platforms such as the one from Cisco<sup>4</sup> will enable such logical data warehouses to be built in the future.

## 29.5 Typical Functionality of a Data Warehouse

Data warehouses exist to facilitate complex, data-intensive, and frequent ad hoc queries. Accordingly, data warehouses must provide far greater and more efficient query support than is demanded of transactional databases. The data warehouse access component supports enhanced spreadsheet functionality, efficient query processing, structured queries, ad hoc queries, data mining, and materialized views. In particular, enhanced spreadsheet functionality includes support for state-of-the-art spreadsheet applications (for example, MS Excel) as well as for OLAP applications programs. These enhanced spreadsheet products offer preprogrammed functionalities such as the following:

- **Roll-up (also drill-up).** Data is summarized with increasing generalization (for example, weekly to quarterly to annually).
- **Drill-down.** Increasing levels of detail are revealed (the complement of roll-up).
- **Pivot.** Cross tabulation (also referred to as *rotation*) is performed.
- **Slice and dice.** Projection operations are performed on the dimensions.
- **Sorting.** Data is sorted by ordinal value.
- **Selection.** Data is filtered by value or range.
- **Derived (computed) attributes.** Attributes are computed by operations on stored and derived values.

Because data warehouses are free from the restrictions of the transactional environment, there is an increased efficiency in query processing. Among the tools and techniques used are query transformation; index intersection and union; special **ROLAP** (relational OLAP) and **MOLAP** (multidimensional OLAP) functions; SQL extensions; advanced join methods; and intelligent scanning (as in piggy-backing multiple queries).

There is also a **HOLAP** (hybrid OLAP) option available that combines both ROLAP and MOLAP. For summary-type information, HOLAP leverages cube technology (using MOLAP) for faster performance. When detailed information is needed, HOLAP can “drill through” from the cube into the underlying relational data (which is in the ROLAP component).

---

<sup>4</sup>See the description of Cisco's Data Virtualization Platform at <http://www.compositesw.com/products-services/data-virtualization-platform/>



Improved performance has also been attained with parallel processing. Parallel server architectures include symmetric multiprocessor (SMP), cluster, and massively parallel processing (MPP), and combinations of these.

Knowledge workers and decision makers use tools ranging from parametric queries to ad hoc queries to data mining. Thus, the access component of the data warehouse must provide support for structured queries (both parametric and ad hoc). Together, these make up a managed query environment. Data mining itself uses techniques from statistical analysis and artificial intelligence. Statistical analysis can be performed by advanced spreadsheets, by sophisticated statistical analysis software, or by custom-written programs. Techniques such as lagging, moving averages, and regression analysis are also commonly employed. Artificial intelligence techniques, which may include genetic algorithms and neural networks, are used for classification and are employed to discover knowledge from the data warehouse that may be unexpected or difficult to specify in queries. (We discussed data mining in detail in Chapter 28.)

## 29.6 Data Warehouse versus Views

Some people consider data warehouses to be an extension of database views. Earlier we mentioned materialized views as one way of meeting requirements for improved access to data (see Section 7.3 for a discussion of views). Materialized views have been explored for their performance enhancement. In Section 19.2.4, we discussed how materialized views are maintained and used as a part of query optimization. Views, however, provide only a subset of the functions and capabilities of data warehouses. Views and data warehouses are similar in some aspects; for example, they both have read-only extracts from databases and they allow orientation by subject. However, data warehouses are different from views in the following ways:

- Data warehouses exist as persistent storage instead of being materialized on demand.
- Data warehouses are not just relational views; they are multidimensional views with levels of aggregation.
- Data warehouses can be indexed to optimize performance. Views cannot be indexed independent of the underlying databases.
- Data warehouses characteristically provide specific support of functionality; views cannot.
- Data warehouses provide large amounts of integrated and often temporal data, generally more than is contained in one database, whereas views are an extract of a database.
- Data warehouses bring in data from multiple sources via a complex ETL process that involves cleaning, pruning, and summarization, whereas views are an extract from a database through a predefined query.

## 29.7 Difficulties of Implementing Data Warehouses

Some significant operational issues arise with data warehousing: construction, administration, and quality control. Project management—the design, construction, and implementation of the warehouse—is an important and challenging consideration that should not be underestimated. The building of an enterprise-wide warehouse in a large organization is a major undertaking, potentially taking years from conceptualization to implementation. Because of the difficulty and amount of lead time required for such an undertaking, the widespread development and deployment of data marts may provide an attractive alternative, especially to those organizations with urgent needs for OLAP, DSS, and/or data mining support.

The administration of a data warehouse is an intensive enterprise, proportional to the size and complexity of the warehouse. An organization that attempts to administer a data warehouse must realistically understand the complex nature of its administration. Although designed for read access, a data warehouse is no more a static structure than any of its information sources. Source databases can be expected to evolve. The warehouse's schema and acquisition component must be expected to be updated to handle these evolutions.

A significant issue in data warehousing is the quality control of data. Both quality and consistency of data—especially as it relates to dimension data, which in turn affects master data management—are major concerns. Although the data passes through a cleaning function during acquisition, quality and consistency remain significant issues for the database administrator and designer alike. Melding data from heterogeneous and disparate sources is a major challenge given differences in naming, domain definitions, identification numbers, and the like. Every time a source database changes, the data warehouse administrator must consider the possible interactions with other elements of the warehouse.

Usage projections should be estimated conservatively prior to construction of the data warehouse and should be revised continually to reflect current requirements. As utilization patterns become clear and change over time, storage and access paths can be tuned to remain optimized for support of the organization's use of its warehouse. This activity should continue throughout the life of the warehouse in order to remain ahead of the demand. The warehouse should also be designed to accommodate the addition and attrition of data sources without major redesign. Sources and source data will evolve, and the warehouse must accommodate such change. Fitting the available source data into the data model of the warehouse will be a continual challenge, a task that is as much art as science. Because there is continual rapid change in technologies, both the requirements and capabilities of the warehouse will change considerably over time. Additionally, data warehousing technology itself will continue to evolve for some time, so component structures and functionalities will continually be upgraded. This certain change is an excellent motivation for fully modular design of components.

Administration of a data warehouse will require far broader skills than are needed for traditional database administration. Often, different parts of a large organization view the data differently. A team of highly skilled technical experts with overlapping areas of expertise will likely be needed, rather than a single individual. The team must also possess a thorough knowledge of the business and specifically the rules and regulations, the constraints and the policies of the enterprise. Like database administration, data warehouse administration is only partly technical; a large part of the responsibility requires working effectively with all the members of the organization who have an interest in the data warehouse. However difficult that can be at times for database administrators, it is that much more challenging for data warehouse administrators because the scope of their responsibilities is considerably broader than that faced by database administrators.

Design of the management function and selection of the management team for a database warehouse are crucial. Managing the data warehouse in a large organization will surely be a major task. Many commercial tools are available to support management functions. Effective data warehouse management will be a team function that requires a wide set of technical skills, careful coordination, and effective leadership. Just as we must prepare for the evolution of the warehouse, we must also recognize that the skills of the management team will, of necessity, evolve with it.

## 29.8 Summary

In this chapter, we surveyed the field known as data warehousing. Data warehousing can be seen as a process that requires a variety of activities to precede it. In contrast, data mining (see Chapter 28) may be thought of as an activity that draws knowledge from an existing data warehouse or other sources of data. We first introduced in Section 29.1 key concepts related to a data warehouse and defined terms such as *OLAP* and *DSS* and contrasted them with *OLTP*. We presented a general architecture of data warehousing systems. We discussed in Section 29.2 the fundamental characteristics of data warehouses and their different types. We then discussed in Section 29.3 the modeling of data in warehouses using what is popularly known as the multidimensional data model. Different types of tables and schemas were discussed. We gave an elaborate account of the processes and design considerations involved in building a data warehouse in Section 29.4. We then presented the typical special functionality associated with a data warehouse in Section 29.5. The view concept from the relational model was contrasted with the multidimensional view of data in data warehouses in Section 29.6. We finally discussed in Section 29.7 the difficulties of implementing data warehouses and the challenges of data warehouse administration.

## Review Questions

**29.1.** What is a data warehouse? How does it differ from a database?

**29.2.** Define the following terms: *OLAP* (online analytical processing), *ROLAP* (relational OLAP), *MOLAP* (multidimensional OLAP), and *DSS* (decision-support systems).

- 29.3. Describe the characteristics of a data warehouse. Divide them into the functionality of a warehouse and the advantages users derive from the warehouse.
- 29.4. What is the multidimensional data model? How is it used in data warehousing?
- 29.5. Define the following terms: *star schema*, *snowflake schema*, *fact constellation*, *data marts*.
- 29.6. What types of indexes are built for a warehouse? Illustrate the uses for each with an example.
- 29.7. Describe the steps of building a warehouse.
- 29.8. What considerations play a major role in the design of a warehouse?
- 29.9. Describe the functions a user can perform on a data warehouse, and illustrate the results of these functions on a sample multidimensional data warehouse.
- 29.10. How is the relational *view* concept similar to a data warehouse and how are they different?
- 29.11. List the difficulties in implementing a data warehouse.
- 29.12. List the ongoing issues and research problems pertaining to data warehousing.
- 29.13. What is master data management? How is it related to data warehousing?
- 29.14. What are logical data warehouses? Do an online search for the data virtualization platform from Cisco, and discuss how it will help in building a logical data warehouse?

## Selected Bibliography

Inmon (1992, 2005) is credited for giving the term wide acceptance. Codd and Salley (1993) popularized the term *online analytical processing* (OLAP) and defined a set of characteristics for data warehouses to support OLAP. Kimball (1996) is known for his contribution to the development of the data warehousing field. Mattison (1996) is one of the several books on data warehousing that gives a comprehensive analysis of techniques available in data warehouses and the strategies companies should use in deploying them. Ponniah (2010) gives a very good practical overview of the data warehouse building process from requirements collection to deployment maintenance. Jukic et al. (2013) is a good source on modeling a data warehouse. Bischoff and Alexander (1997) is a compilation of advice from experts. Chaudhuri and Dayal (1997) give an excellent tutorial on the topic, while Widom (1995) points to a number of ongoing issues and research.