INSTRUCTOR'S MANUAL TO ACCOMPANY

Database System Concepts

Fourth Edition

Abraham Silberschatz
Yale University

Henry F. Korth
Lehigh University

S. Sudarshan
Indian Institute of Technology, Bombay

Contents

Preface

This volume is an instructor's manual for the 4^{th} edition of *Database System Concepts* by Abraham Silberschatz, Henry F. Korth and S. Sudarshan. It contains answers to the exercises at the end of each chapter of the book. Before providing answers to the exercises for each chapter, we include a few remarks about the chapter. The nature of these remarks vary. They include explanations of the inclusion or omission of certain material, and remarks on how we teach the chapter in our own courses. The remarks also include suggestions on material to skip if time is at a premium, and tips on software and supplementary material that can be used for programming exercises.

Beginning with this edition, solutions for some problems have been made available on the Web. These problems have been marked with a " * " in the instructor's manual.

The Web home page of the book, at http://www.db-book.com, contains a variety of useful information, including up-to-date errata, online appendices describing the network data model, the hierarchical data model, and advanced relational database design, and model course syllabi. We will periodically update the page with supplementary material that may be of use to teachers and students.

We provide a mailing list through which users can communicate among themselves and with us. If you wish to use this facility, please visit the following URL and follow the instructions there to subscribe:

http://mailman.cs.yale.edu/mailman/listinfo/db-book-list

The mailman mailing list system provides many benefits, such as an archive of postings, and several subscription options, including digest and Web only. To send messages to the list, send e-mail to:

db-book-list@cs.yale.edu

We would appreciate it if you would notify us of any errors or omissions in the book, as well as in the instructor's manual. Internet electronic mail should be ad-

dressed to db-book@cs.yale.edu. Physical mail may be sent to Avi Silberschatz, Yale University, 51 Prospect Street, New Haven, CT, 06520, USA.

Although we have tried to produce an instructor's manual which will aid all of the users of our book as much as possible, there can always be improvements. These could include improved answers, additional questions, sample test questions, programming projects, suggestions on alternative orders of presentation of the material, additional references, and so on. If you would like to suggest any such improvements to the book or the instructor's manual, we would be glad to hear from you. All contributions that we make use of will, of course, be properly credited to their contributor.

Nilesh Dalvi, Sumit Sanghai, Gaurav Bhalotia and Arvind Hulgeri did the bulk of the work in preparing the instructors manual for the 4^{th} edition. This manual is derived from the manuals for the earlier editions. The manual for the 3^{rd} edition was prepared by K. V. Raghavan with help from Prateek R. Kapadia, Sara Strandtman helped with the instructor manual for the 2^{nd} and 3^{rd} editions, while Greg Speegle and Dawn Bezviner helped us to prepare the instructor's manual for the 1^{st} edition.

A. S.
H. F. K.
S. S.

Instructor Manual Version 4.0.0

CHAPTER 1

Introduction

Chapter 1 provides a general overview of the nature and purpose of database systems. The most important concept in this chapter is that database systems allow data to be treated at a high level of abstraction. Thus, database systems differ significantly from the file systems and general purpose programming environments with which students are already familiar. Another important aspect of the chapter is to provide motivation for the use of database systems as opposed to application programs built on top of file systems. Thus, the chapter motivates what the student will be studying in the rest of the course.

The idea of abstraction in database systems deserves emphasis throughout, not just in discussion of Section 1.3. The overview of the structure of databases, starting from Section 1.4 is, of necessity, rather brief, and is meant only to give the student a rough idea of some of the concepts. The student may not initially be able to fully appreciate the concepts described here, but should be able to do so by the end of the course.

The specifics of the E-R, relational, and object-oriented models are covered in later chapters. These models can be used in Chapter 1 to reinforce the concept of abstraction, with syntactic details deferred to later in the course.

If students have already had a course in operating systems, it is worthwhile to point out how the OS and DBMS are related. It is useful also to differentiate between concurrency as it is taught in operating systems courses (with an orientation towards files, processes, and physical resources) and database concurrency control (with an orientation towards granularity finer than the file level, recoverable transactions, and resources accessed associatively rather than physically). If students are familiar with a particular operating system, that OS's approach to concurrent file access may be used for illustration.

Exercises

1.1 List four significant differences between a file-processing system and a DBMS.
Answer: Some main differences between a database management system and a file-processing system are:

- Both systems contain a collection of data and a set of programs which access that data. A database management system coordinates both the physical and the logical access to the data, whereas a file-processing system coordinates only the physical access.
- A database management system reduces the amount of data duplication by ensuring that a physical piece of data is available to all programs authorized to have access to it, whereas data written by one program in a file-processing system may not be readable by another program.
- A database management system is designed to allow flexible access to data (i.e., queries), whereas a file-processing system is designed to allow predetermined access to data (i.e., compiled programs).
- A database management system is designed to coordinate multiple users accessing the same data at the same time. A file-processing system is usually designed to allow one or more programs to access different data files at the same time. In a file-processing system, a file can be accessed by two programs concurrently only if both programs have read-only access to the file.

1.2 This chapter has described several major advantages of a database system. What are two disadvantages?
Answer: Two disadvantages associated with database systems are listed below.

a. Setup of the database system requires more knowledge, money, skills, and time.
b. The complexity of the database may result in poor performance.

1.3 Explain the difference between physical and logical data independence.
Answer:

- Physical data independence is the ability to modify the physical scheme without making it necessary to rewrite application programs. Such modifications include changing from unblocked to blocked record storage, or from sequential to random access files.
- Logical data independence is the ability to modify the conceptual scheme without making it necessary to rewrite application programs. Such a modification might be adding a field to a record; an application program's view hides this change from the program.

1.4 List five responsibilities of a database management system. For each responsibility, explain the problems that would arise if the responsibility were not discharged.
Answer: A general purpose database manager (DBM) has five responsibilities:

a. interaction with the file manager.

b. integrity enforcement.
c. security enforcement.
d. backup and recovery.
e. concurrency control.

If these responsibilities were not met by a given DBM (and the text points out that sometimes a responsibility is omitted by design, such as concurrency control on a single-user DBM for a micro computer) the following problems can occur, respectively:

a. No DBM can do without this, if there is no file manager interaction then nothing stored in the files can be retrieved.
b. Consistency constraints may not be satisfied, account balances could go below the minimum allowed, employees could earn too much overtime (e.g., hours > 80) or, airline pilots may fly more hours than allowed by law.
c. Unauthorized users may access the database, or users authorized to access part of the database may be able to access parts of the database for which they lack authority. For example, a high school student could get access to national defense secret codes, or employees could find out what their supervisors earn.
d. Data could be lost permanently, rather than at least being available in a consistent state that existed prior to a failure.
e. Consistency constraints may be violated despite proper integrity enforcement in each transaction. For example, incorrect bank balances might be reflected due to simultaneous withdrawals and deposits, and so on.

1.5 What are five main functions of a database administrator?
Answer: Five main functions of a database administrator are:

- To create the scheme definition
- To define the storage structure and access methods
- To modify the scheme and/or physical organization when necessary
- To grant authorization for data access
- To specify integrity constraints

1.6 List seven programming languages that are procedural and two that are non-procedural. Which group is easier to learn and use? Explain your answer.
Answer: Programming language classification:

- Procedural: C, C++, Java, Basic, Fortran, Cobol, Pascal
- Non-procedural: Lisp and Prolog

Note: Lisp and Prolog support some procedural constructs, but the core of both these languages is non-procedural.

In theory, non-procedural languages are easier to learn, because they let the programmer concentrate on *what* needs to be done, rather than *how* to do it. This is not always true in practice, especially if procedural languages are learned first.

1.7 List six major steps that you would take in setting up a database for a particular enterprise.

Answer: Six major steps in setting up a database for a particular enterprise are:

- Define the high level requirements of the enterprise (this step generates a document known as the system requirements specification.)
- Define a model containing all appropriate types of data and data relationships.
- Define the integrity constraints on the data.
- Define the physical level.
- For each known problem to be solved on a regular basis (e.g., tasks to be carried out by clerks or Web users) define a user interface to carry out the task, and write the necessary application programs to implement the user interface.
- Create/initialize the database.

1.8 Consider a two-dimensional integer array of size $n \times m$ that is to be used in your favorite programming language. Using the array as an example, illustrate the difference (a) between the three levels of data abstraction, and (b) between a schema and instances.

Answer: Let *tgrid* be a two-dimensional integer array of size $n \times m$.

a.
- The physical level would simply be $m \times n$ (probably consecutive) storage locations of whatever size is specified by the implementation (e.g., 32 bits each).
- The conceptual level is a grid of boxes, each possibly containing an integer, which is *n* boxes high by *m* boxes wide.
- There are $2^{m \times n}$ possible views. For example, a view might be the entire array, or particular row of the array, or all *n* rows but only columns 1 through *i*.

b.
- Consider the following Pascal declarations:

  type *tgrid* = **array**[1..*n*, 1..*m*] **of integer**;
  var *vgrid1*, *vgrid2* : *tgrid*

  Then *tgrid* is a schema, whereas the value of variables *vgrid1* and *vgrid2* are instances.
- To illustrate further, consider the schema **array**[1..2, 1..2] **of integer**. Two instances of this scheme are:

1	16
7	89

17	90
412	8

CHAPTER 2

Entity Relationship Model

This chapter introduces the entity-relationship model in detail. The chapter covers numerous features of the model, several of which can be omitted depending on the planned coverage of the course. Weak entity sets (Section 2.6), design constraints (Section 2.7.4) and aggregation (Section 2.7.5), and the corresponding subsections of Section 2.9 (Reduction of an E-R Schema to Tables) can be omitted if time is short. We recommend covering specialization (Section 2.7.1) at least in some detail, since it is an important concept for object-oriented databases (Chapter 8).

The E-R model itself and E-R diagrams are used often in the text. It is important that students become comfortable with them. The E-R model is an excellent context for the introduction of students to the complexity of database design. For a given enterprise there are often a wide variety of E-R designs. Although some choices are arbitrary, it is often the case that one design is inherently superior to another. Several of the exercises illustrate this point. The evaluation of the goodness of an E-R design requires an understanding of the enterprise being modeled and the applications to be run. It is often possible to lead students into a debate of the relative merits of competing designs and thus illustrate by example that understanding the application is often the hardest part of database design.

Considerable emphasis is placed on the construction of tables from E-R diagrams. This serves to build intuition for the discussion of the relational model in the subsequent chapters. It also serves to ground abstract concepts of entities and relationships into the more concrete concepts of relations. Several other texts places this material along with the relational data model, rather than in the E-R model chapter. Our motivation for placing this material here is help students to appreciate how E-R data models get used in reality, while studying the E-R model rather than later on.

The material on conversion of E-R diagrams to tables in the book is rather brief in some places, the book slides provide better coverage of details that have been left implicit in the book.

Changes from 3rd edition:

In the fourth edition we have updated several examples, including ternary relations (*employee, branch, job* instead of *customer, loan, branch*) and aggregation (*manages* instead of *loan-officer*), to make them more realistic. We have also added more examples, for instance for specialization we use *person, customer* and *employee* as the main example, instead of *account, checking-account* and *savings-account*, which also makes the example more realistic. We have replaced the US centric *social-security* by the more global (and more realistic) *customer-id* and *employee-id*.

We have added notation to make disjointedness constraints and total participation explicit (overlapping and partial participation are the default). We have introduced alternative E-R notations since many real world applications use alternative notations. We have also provided a brief introduction to UML class diagrams, which are being used increasingly in place of E-R diagrams, in tools such as Oracle designer.

We have dropped coverage of existence dependencies since total participation constraints provide a very similar constraint. The distinction between total participation and existence dependencies is too minor to be of practical use, and only confuses students.

Design issues are discussed in more detail.

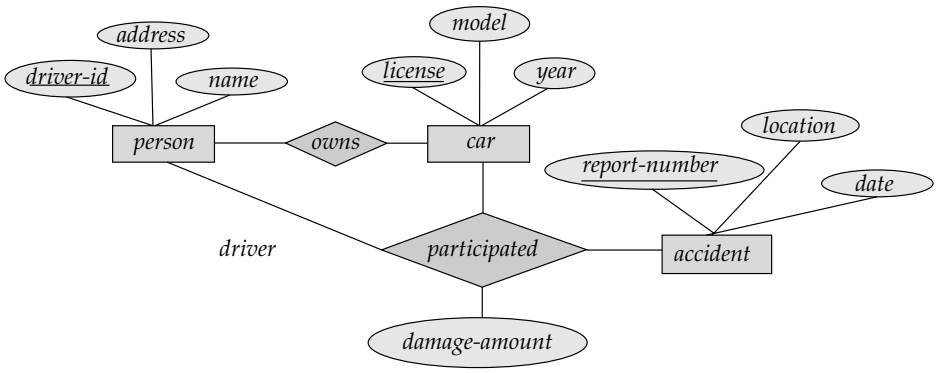


Figure 2.1 E-R diagram for a Car-insurance company.

Exercises

2.1 Explain the distinctions among the terms primary key, candidate key, and superkey.

Answer: A *superkey* is a set of one or more attributes that, taken collectively, allows us to identify uniquely an entity in the entity set. A superkey may contain extraneous attributes. If K is a superkey, then so is any superset of K. A superkey for which no proper subset is also a superkey is called a *candidate key*. It is possible that several distinct sets of attributes could serve as candidate keys. The *primary key* is one of the candidate keys that is chosen by the database designer as the principal means of identifying entities within an entity set.

2.2 Construct an E-R diagram for a car-insurance company whose customers own one or more cars each. Each car has associated with it zero to any number of recorded accidents.

Answer: See Figure 2.1

2.3 Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.

Answer: See Figure 2.2

2.4 A university registrar's office maintains data about the following entities: (a) courses, including number, title, credits, syllabus, and prerequisites; (b) course offerings, including course number, year, semester, section number, instructor(s), timings, and classroom; (c) students, including student-id, name, and program; and (d) instructors, including identification number, name, department, and title. Further, the enrollment of students in courses and grades awarded to students in each course they are enrolled for must be appropriately modeled.

Construct an E-R diagram for the registrar's office. Document all assumptions that you make about the mapping constraints.

Answer: See Figure 2.3.

In the answer given here, the main entity sets are *student*, *course*, *course-offering*,

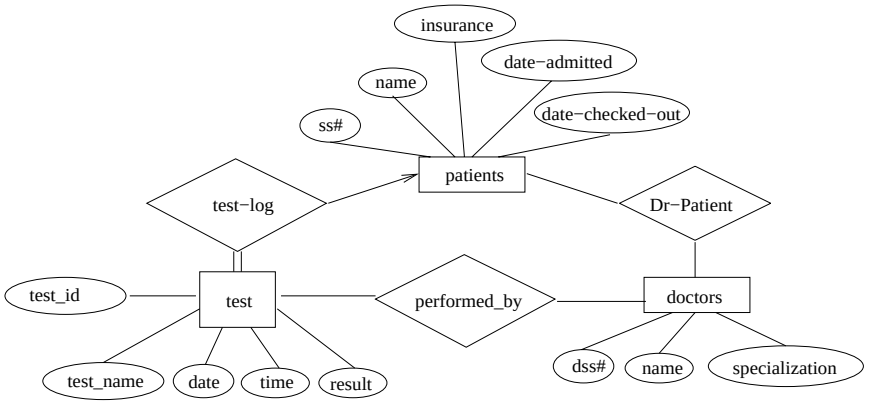


Figure 2.2 E-R diagram for a hospital.

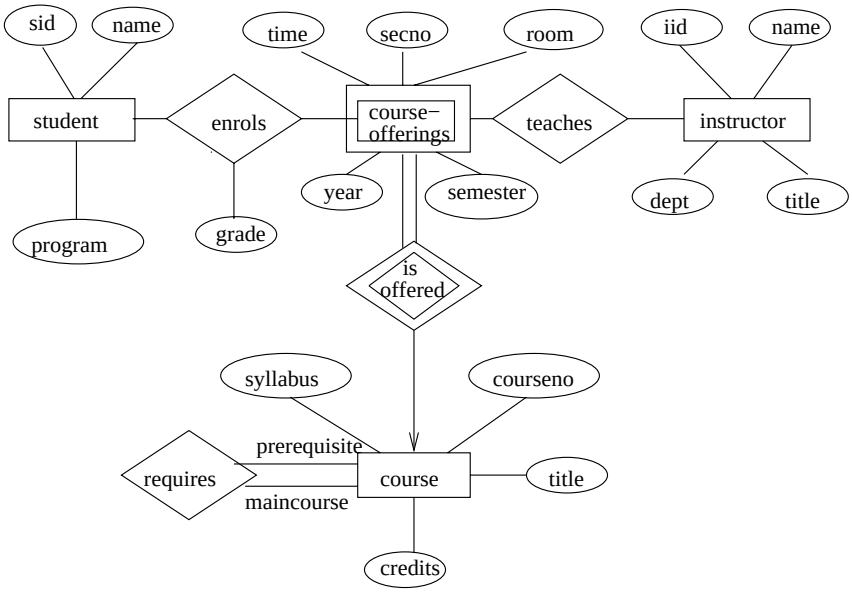


Figure 2.3 E-R diagram for a university.

and *instructor*. The entity set *course-offering* is a weak entity set dependent on *course*. The assumptions made are :

a. a class meets only at one particular place and time. This E-R diagram cannot model a class meeting at different places at different times.

b. There is no guarantee that the database does not have two classes meeting at the same place and time.

2.5 Consider a database used to record the marks that students get in different exams of different course offerings.

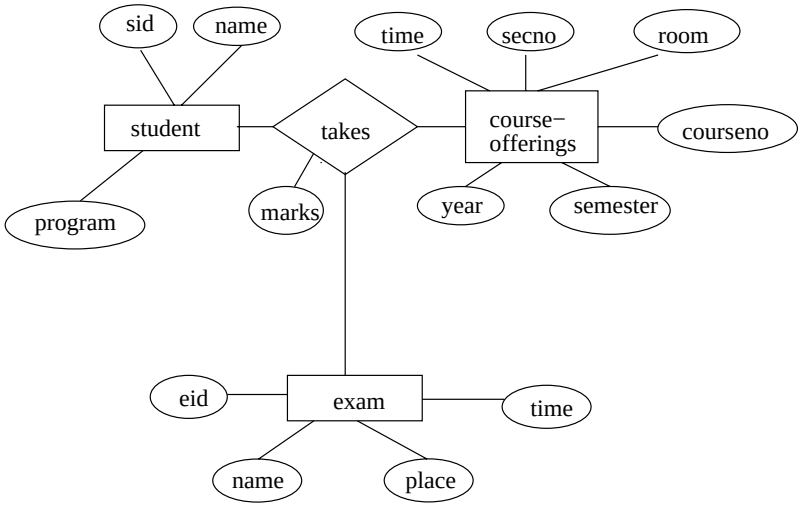


Figure 2.4 E-R diagram for marks database.

a. Construct an E-R diagram that models exams as entities, and uses a ternary relationship, for the above database.

b. Construct an alternative E-R diagram that uses only a binary relationship between *students* and *course-offerings*. Make sure that only one relationship exists between a particular student and course-offering pair, yet you can represent the marks that a student gets in different exams of a course offering.

Answer:

a. See Figure 2.4

b. See Figure 2.5

2.6 Construct appropriate tables for each of the E-R diagrams in Exercises 2.2 to 2.4.

Answer:

a. Car insurance tables:

person (driver-id, name, address)
car (license, year, model)
accident (report-number, date, location)
participated(driver-id, license, report-number, damage-amount)

b. Hospital tables:

patients (patient-id, name, insurance, date-admitted, date-checked-out)
doctors (doctor-id, name, specialization)
test (testid, testname, date, time, result)
doctor-patient (patient-id, doctor-id)
test-log (testid, patient-id) performed-by (testid, doctor-id)

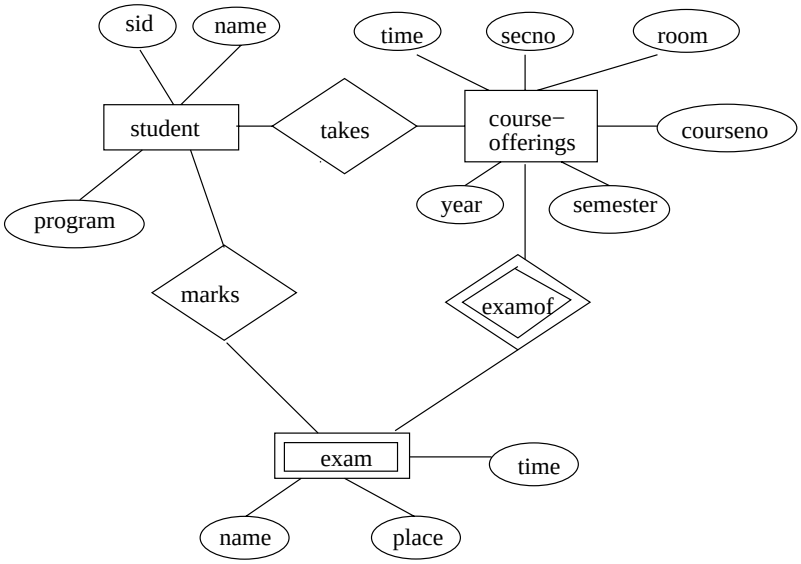


Figure 2.5 Another E-R diagram for marks database.

c. University registrar's tables:

student (<u>student-id</u>, name, program)
course (<u>courseno</u>, title, syllabus, credits)
course-offering (<u>courseno</u>, <u>secno</u>, <u>year</u>, <u>semester</u>, time, room)
instructor (<u>instructor-id</u>, name, dept, title)
enrols (<u>student-id</u>, <u>courseno</u>, <u>secno</u>, <u>semester</u>, <u>year</u>, grade)
teaches (<u>courseno</u>, <u>secno</u>, <u>semester</u>, <u>year</u>, <u>instructor-id</u>)
requires (<u>maincourse</u>, <u>prerequisite</u>)

2.7 Design an E-R diagram for keeping track of the exploits of your favourite sports team. You should store the matches played, the scores in each match, the players in each match and individual player statistics for each match. Summary statistics should be modeled as derived attributes.
Answer: See Figure 2.6

2.8 Extend the E-R diagram of the previous question to track the same information for all teams in a league.
Answer: See Figure 2.7 Note that a player can stay in only one team during a season.

2.9 Explain the difference between a weak and a strong entity set.
Answer: A strong entity set has a primary key. All tuples in the set are distinguishable by that key. A weak entity set has no primary key unless attributes of the strong entity set on which it depends are included. Tuples in a weak entity set are partitioned according to their relationship with tuples in a strong entity

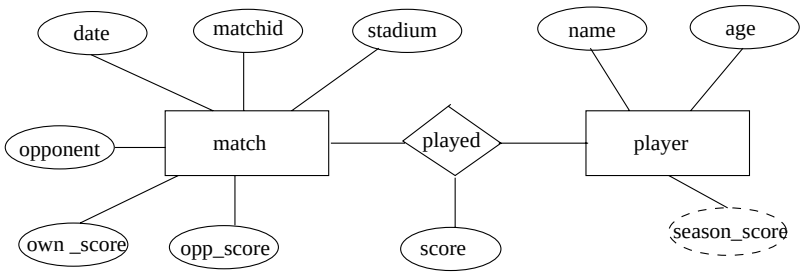


Figure 2.6 E-R diagram for favourite team statistics.

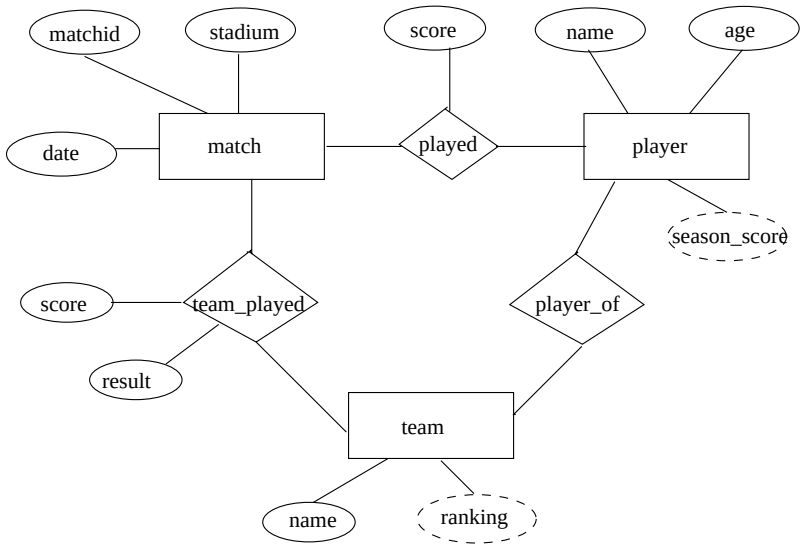


Figure 2.7 E-R diagram for all teams statistics.

set. Tuples within each partition are distinguishable by a discriminator, which is a set of attributes.

2.10 We can convert any weak entity set to a strong entity set by simply adding appropriate attributes. Why, then, do we have weak entity sets?
Answer: We have weak entities for several reasons:

- We want to avoid the data duplication and consequent possible inconsistencies caused by duplicating the key of the strong entity.
- Weak entities reflect the logical structure of an entity being dependent on another entity.
- Weak entities can be deleted automatically when their strong entity is deleted.
- Weak entities can be stored physically with their strong entities.

2.11 Define the concept of aggregation. Give two examples of where this concept is useful.

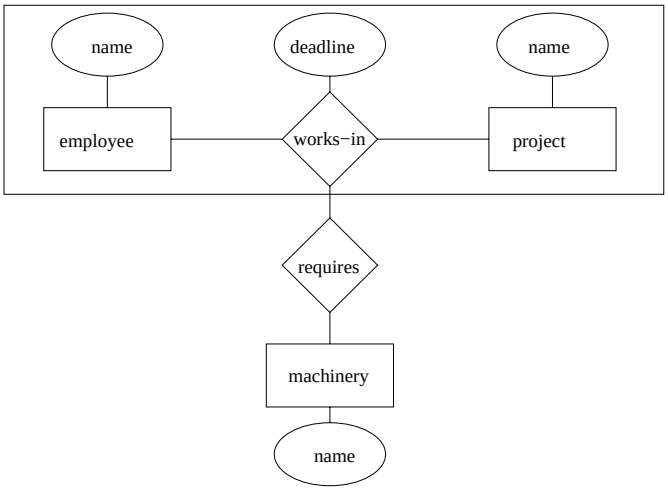


Figure 2.8 E-R diagram Example 1 of aggregation.

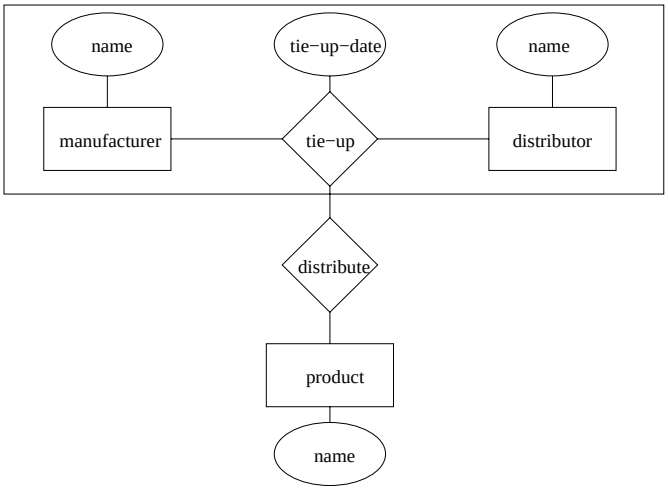


Figure 2.9 E-R diagram Example 2 of aggregation.

Answer: Aggregation is an abstraction through which relationships are treated as higher-level entities. Thus the relationship between entities A and B is treated as if it were an entity C. Some examples of this are:

- **a.** Employees work for projects. An employee working for a particular project uses various machinery. See Figure 2.8
- **b.** Manufacturers have tie-ups with distributors to distribute products. Each tie-up has specified for it the set of products which are to be distributed. See Figure 2.9

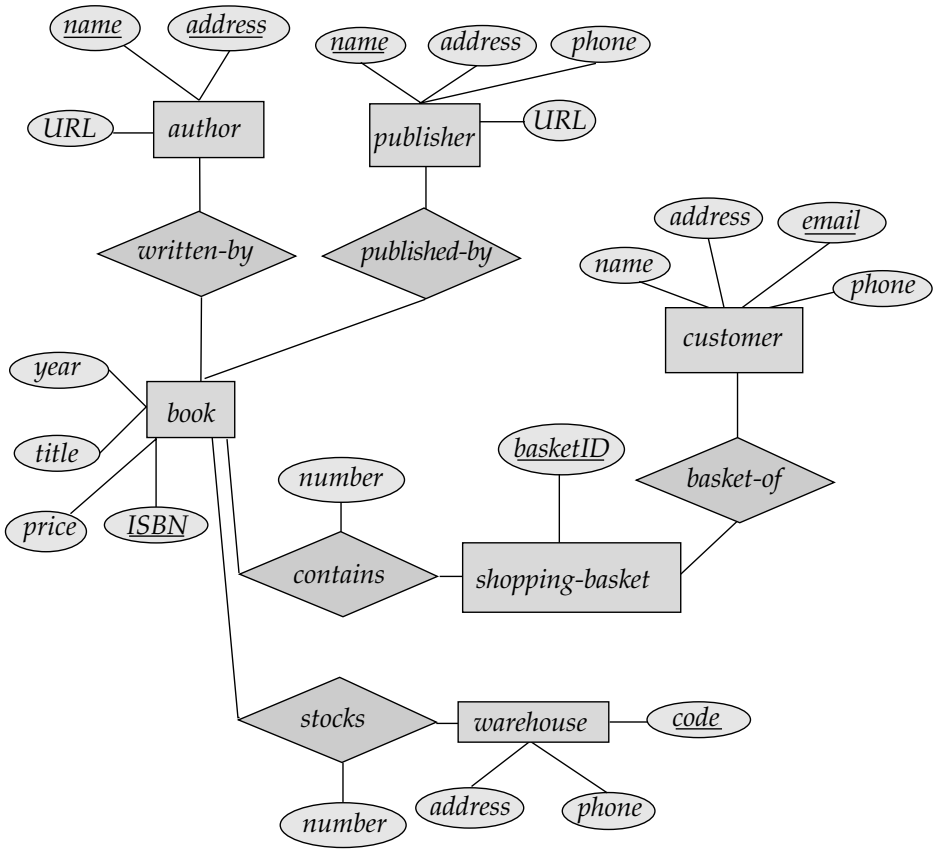


Figure 2.10 E-R diagram for Exercise 2.12.

2.12 Consider the E-R diagram in Figure 2.10, which models an online bookstore.

- **a.** List the entity sets and their primary keys.
- **b.** Suppose the bookstore adds music cassettes and compact disks to its collection. The same music item may be present in cassette or compact disk format, with differing prices. Extend the E-R diagram to model this addition, ignoring the effect on shopping baskets.
- **c.** Now extend the E-R diagram, using generalization, to model the case where a shopping basket may contain any combination of books, music cassettes, or compact disks.

Answer:

2.13 Consider an E-R diagram in which the same entity set appears several times. Why is allowing this redundancy a bad practice that one should avoid whenever possible?

Answer: By using one entity set many times we are missing relationships in

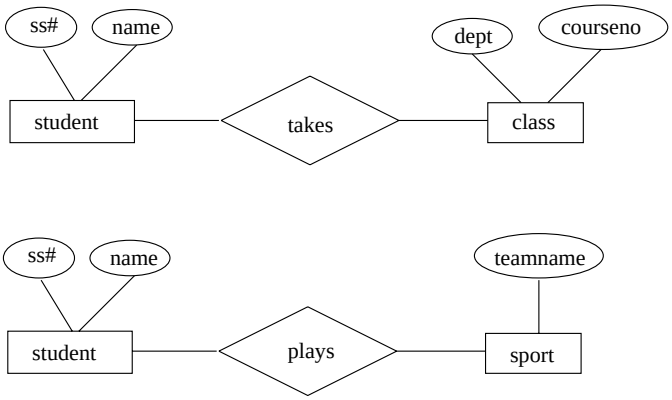


Figure 2.11 E-R diagram with entity duplication.

the model. For example, in the E-R diagram in Figure 2.11: the students taking classes are the same students who are athletes, but this model will not show that.

2.14 Consider a university database for the scheduling of classrooms for final exams. This database could be modeled as the single entity set *exam*, with attributes *course-name*, *section-number*, *room-number*, and *time*. Alternatively, one or more additional entity sets could be defined, along with relationship sets to replace some of the attributes of the *exam* entity set, as

- *course* with attributes *name*, *department*, and *c-number*
- *section* with attributes *s-number* and *enrollment*, and dependent as a weak entity set on *course*
- *room* with attributes *r-number*, *capacity*, and *building*

a. Show an E-R diagram illustrating the use of all three additional entity sets listed.

b. Explain what application characteristics would influence a decision to include or not to include each of the additional entity sets.

Answer:

a. See Figure 2.12

b. The additional entity sets are useful if we wish to store their attributes as part of the database. For the *course* entity set, we have chosen to include three attributes. If only the primary key (*c-number*) were included, and if courses have only one section, then it would be appropriate to replace the *course* (and *section*) entity sets by an attribute (*c-number*) of *exam*. The reason it is undesirable to have multiple attributes of *course* as attributes of *exam* is that it would then be difficult to maintain data on the courses, particularly if a course has no exam or several exams. Similar remarks apply to the *room* entity set.

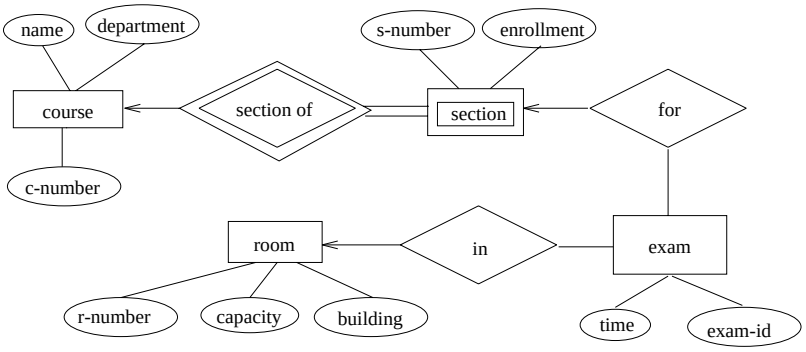


Figure 2.12 E-R diagram for exam scheduling.

2.15 When designing an E-R diagram for a particular enterprise, you have several alternatives from which to choose.

a. What criteria should you consider in making the appropriate choice?

b. Design three alternative E-R diagrams to represent the university registrar's office of Exercise 2.4. List the merits of each. Argue in favor of one of the alternatives.

Answer:

a. The criteria to use are intuitive design, accurate expression of the real-world concept and efficiency. A model which clearly outlines the objects and relationships in an intuitive manner is better than one which does not, because it is easier to use and easier to change. Deciding between an attribute and an entity set to represent an object, and deciding between an entity set and relationship set, influence the accuracy with which the real-world concept is expressed. If the right design choice is not made, inconsistency and/or loss of information will result. A model which can be implemented in an efficient manner is to be preferred for obvious reasons.

b. Consider three different alternatives for the problem in Exercise 2.4.

- See Figure 2.13
- See Figure 2.14
- See Figure 2.15

Each alternative has merits, depending on the intended use of the database. Scheme 2.13 has been seen earlier. Scheme 2.15 does not require a separate entity for *prerequisites*. However, it will be difficult to store all the prerequisites(being a multi-valued attribute). Scheme 2.14 treats prerequisites as well as classrooms as separate entities, making it useful for gathering data about prerequisites and room usage. Scheme 2.13 is in between the others, in that it treats prerequisites as separate entities but not classrooms. Since a registrar's office probably has to answer general questions about the number of classes a student is taking or what are all the prerequisites of a course, or where a specific class meets, scheme 2.14 is probably the best choice.

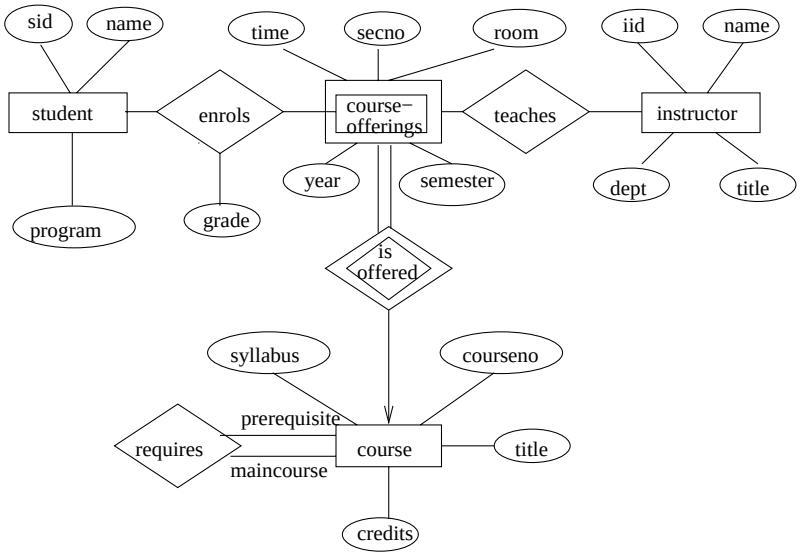


Figure 2.13 E-R diagram for University(a) .

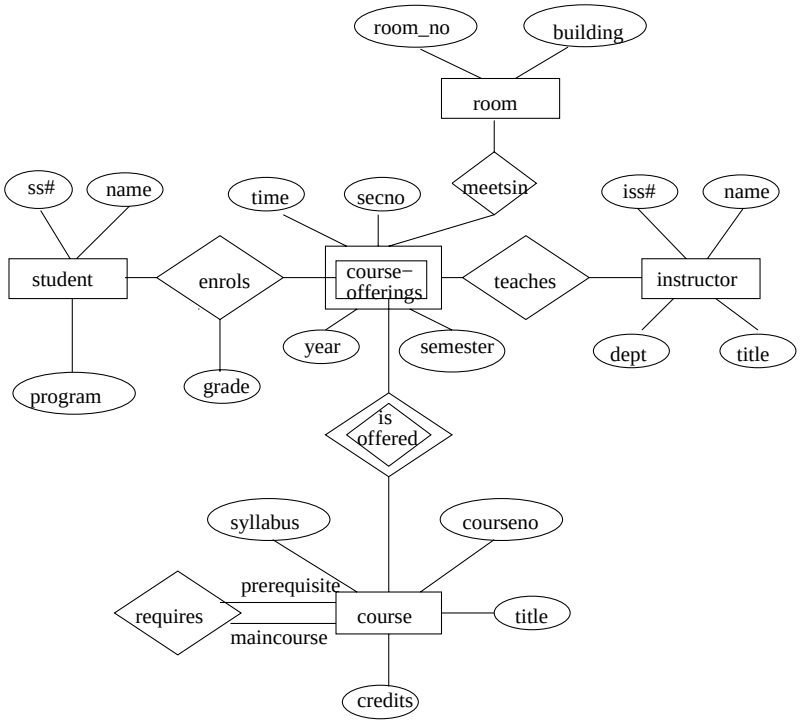


Figure 2.14 E-R diagram for University(b).

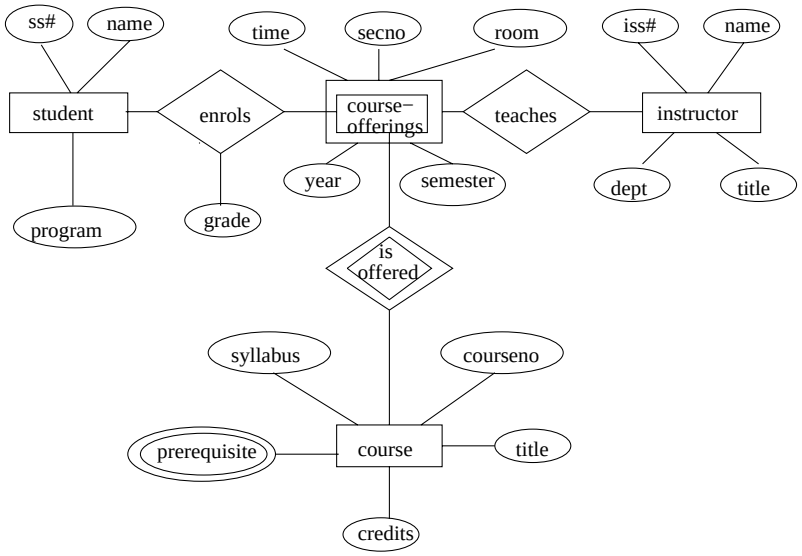


Figure 2.15 E-R diagram for University(c).

2.16 An E-R diagram can be viewed as a graph. What do the following mean in terms of the structure of an enterprise schema?

a. The graph is disconnected.
b. The graph is acyclic.

Answer:

a. If a pair of entity sets are connected by a path in an E-R diagram, the entity sets are related, though perhaps indirectly. A disconnected graph implies that there are pairs of entity sets that are unrelated to each other. If we split the graph into connected components, we have, in effect, a separate database corresponding to each connected component.
b. As indicated in the answer to the previous part, a path in the graph between a pair of entity sets indicates a (possibly indirect) relationship between the two entity sets. If there is a cycle in the graph then every pair of entity sets on the cycle are related to each other in at least two distinct ways. If the E-R diagram is acyclic then there is a unique path between every pair of entity sets and, thus, a unique relationship between every pair of entity sets.

2.17 In Section 2.4.3, we represented a ternary relationship (Figure 2.16a) using binary relationships, as shown in Figure 2.16b. Consider the alternative shown in Figure 2.16c. Discuss the relative merits of these two alternative representations of a ternary relationship by binary relationships.
Answer: The model of Figure 2.16c will not be able to represent all ternary relationships. Consider the *ABC* relationship set below.

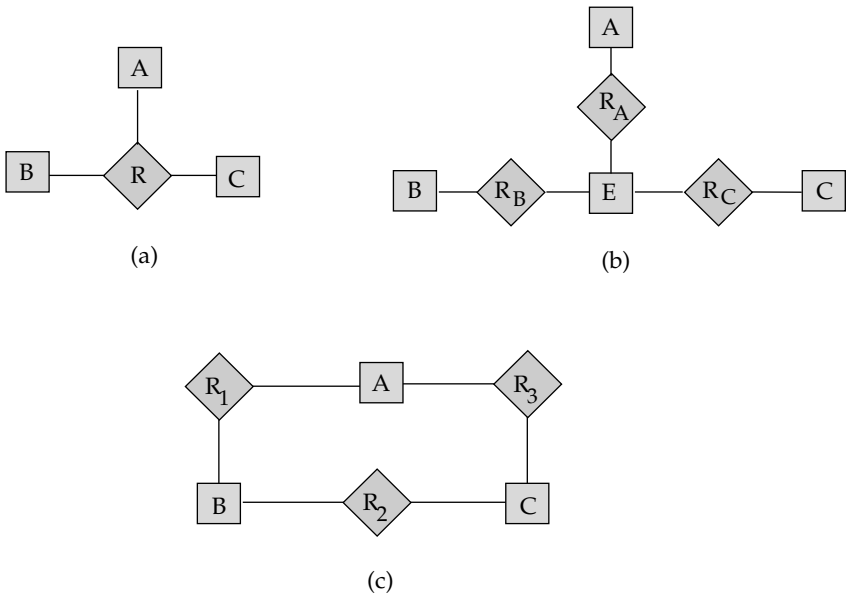


Figure 2.16 E-R diagram for Exercise 2.17 (attributes not shown.)

A	B	C
1	2	3
4	2	7
4	8	3

If ABC is broken into three relationships sets AB, BC and AC, the three will imply that the relation (4, 2, 3) is a part of ABC.

2.18 Consider the representation of a ternary relationship using binary relationships as described in Section 2.4.3 (shown in Figure 2.16b.)

- **a.** Show a simple instance of E, A, B, C, R_A, R_B, and R_C that cannot correspond to any instance of A, B, C, and R.
- **b.** Modify the E-R diagram of Figure 2.16b to introduce constraints that will guarantee that any instance of E, A, B, C, R_A, R_B, and R_C that satisfies the constraints will correspond to an instance of A, B, C, and R.
- **c.** Modify the translation above to handle total participation constraints on the ternary relationship.
- **d.** The above representation requires that we create a primary key attribute for E. Show how to treat E as a weak entity set so that a primary key attribute is not required.

Answer:

- **a.** Let $E = \{e_1, e_2\}$, $A = \{a_1, a_2\}$, $B = \{b_1\}$, $C = \{c_1\}$, $R_A = \{(e_1, a_1), (e_2, a_2)\}$, $R_B = \{(e_1, b_1)\}$, and $R_C = \{(e_1, c_1)\}$. We see that because of the tuple (e_2, a_2), no instance of R exists which corresponds to E, R_A, R_B and R_C.

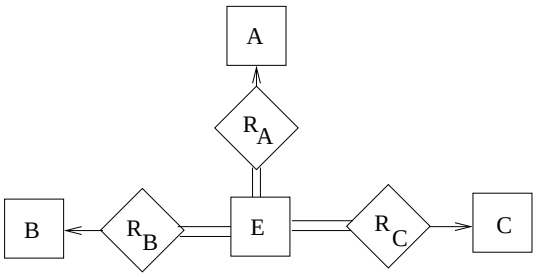


Figure 2.17 E-R diagram to Exercise 2.17b.

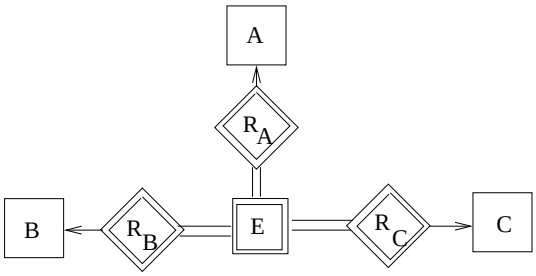


Figure 2.18 E-R diagram to Exercise 2.17d.

b. See Figure 2.17. The idea is to introduce total participation constraints between E and the relationships R_A, R_B, R_C so that every tuple in E has a relationship with A, B and C.

c. Suppose A totally participates in the relationhip R, then introduce a total participation constraint between A and R_A.

d. Consider E as a weak entity set and R_A, R_B and R_C as its identifying relationship sets. See Figure 2.18.

2.19 A weak entity set can always be made into a strong entity set by adding to its attributes the primary key attributes of its identifying entity set. Outline what sort of redundancy will result if we do so.

Answer: The primary key of a weak entity set can be inferred from its relationship with the strong entity set. If we add primary key attributes to the weak entity set, they will be present in both the entity set and the relationship set and they have to be the same. Hence there will be redundancy.

2.20 Design a generalization–specialization hierarchy for a motor-vehicle sales company. The company sells motorcycles, passenger cars, vans, and buses. Justify your placement of attributes at each level of the hierarchy. Explain why they should not be placed at a higher or lower level.

Answer: Figure 2.19 gives one possible hierarchy, there could be many different solutions. The generalization–specialization hierarchy for the motor-vehicle company is given in the figure. *model*, *sales-tax-rate* and *sales-volume* are attributes necessary for all types of vehicles. Commercial vehicles attract commercial vehi-

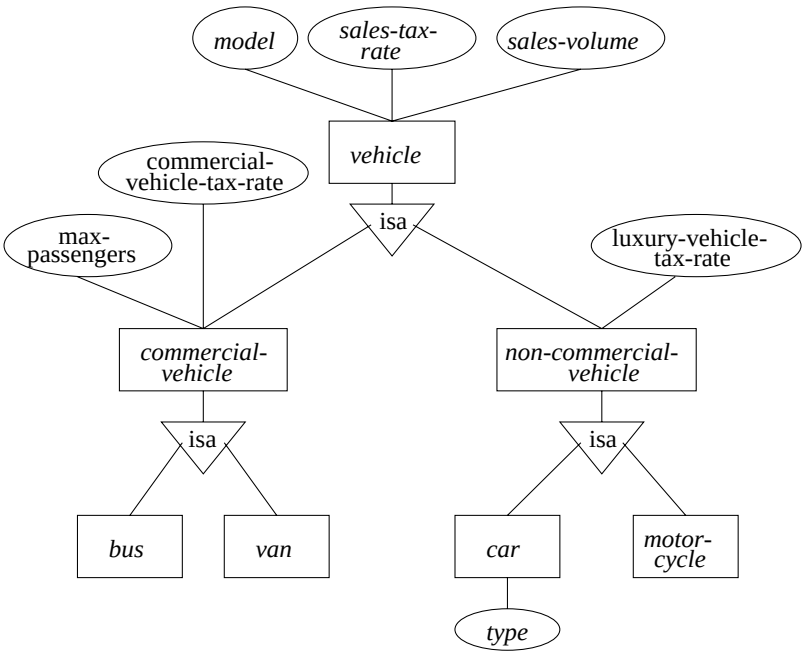


Figure 2.19 E-R diagram of motor-vehicle sales company.

cle tax, and each kind of commercial vehicle has a passenger carrying capacity specified for it. Some kinds of non-commercial vehicles attract luxury vehicle tax. Cars alone can be of several types, such as sports-car, sedan, wagon etc., hence the attribute *type*.

2.21 Explain the distinction between condition-defined and user-defined constraints. Which of these constraints can the system check automatically? Explain your answer.

Answer: In a generalization–specialization hierarchy, it must be possible to decide which entities are members of which lower level entity sets. In a condition-defined design constraint, membership in the lower level entity-sets is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate.User-defined lower-level entity sets are not constrained by a membership condition; rather, entities are assigned to a given entity set by the database user.

Condition-defined constraints alone can be automatically handled by the system. Whenever any tuple is inserted into the database, its membership in the various lower level entity-sets can be automatically decided by evaluating the respective membership predicates. Similarly when a tuple is updated, its membership in the various entity sets can be re-evaluated automatically.

2.22 Explain the distinction between disjoint and overlapping constraints.

Answer: In a disjointness design constraint, an entity can belong to not more

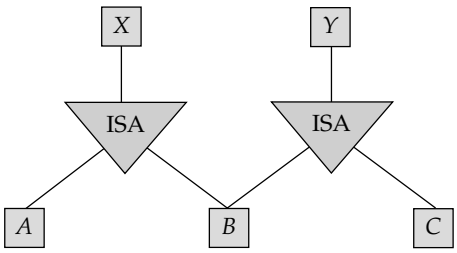


Figure 2.20 E-R diagram for Exercise 2.24 (attributes not shown).

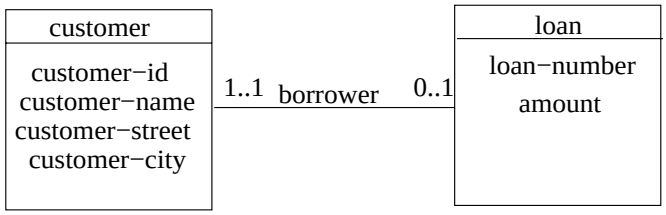


Figure 2.21 UML equivalent of Figure 2.9c.

than one lower-level entity set. In overlapping generalizations, the same entity may belong to more than one lower-level entity sets. For example, in the employee-workteam example of the book, a manager may participate in more than one work-team.

2.23 Explain the distinction between total and partial constraints.
Answer: In a total design constraint, each higher-level entity must belong to a lower-level entity set. The same need not be true in a partial design constraint. For instance, some employees may belong to no work-team.

2.24 Figure 2.20 shows a lattice structure of generalization and specialization. For entity sets A, B, and C, explain how attributes are inherited from the higher-level entity sets X and Y. Discuss how to handle a case where an attribute of X has the same name as some attribute of Y.
Answer: A inherits all the attributes of X plus it may define its own attributes. Similarly C inherits all the attributes of Y plus its own attributes. B inherits the attributes of both X and Y. If there is some attribute *name* which belongs to both X and Y, it may be referred to in B by the qualified name $X.name$ or $Y.name$.

2.25 Draw the UML equivalents of the E-R diagrams of Figures 2.9c, 2.10, 2.12, 2.13 and 2.17.
Answer: See Figures 2.21 to 2.25

2.26 Consider two separate banks that decide to merge. Assume that both banks use exactly the same E-R database schema—the one in Figure 2.22. (This assumption is, of course, highly unrealistic; we consider the more realistic case in

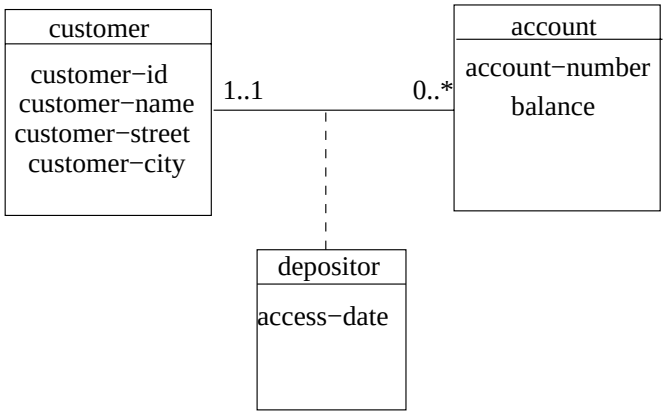


Figure 2.22 UML equivalent of Figure 2.10

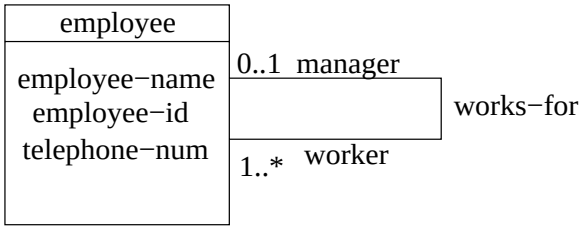


Figure 2.23 UML equivalent of Figure 2.12

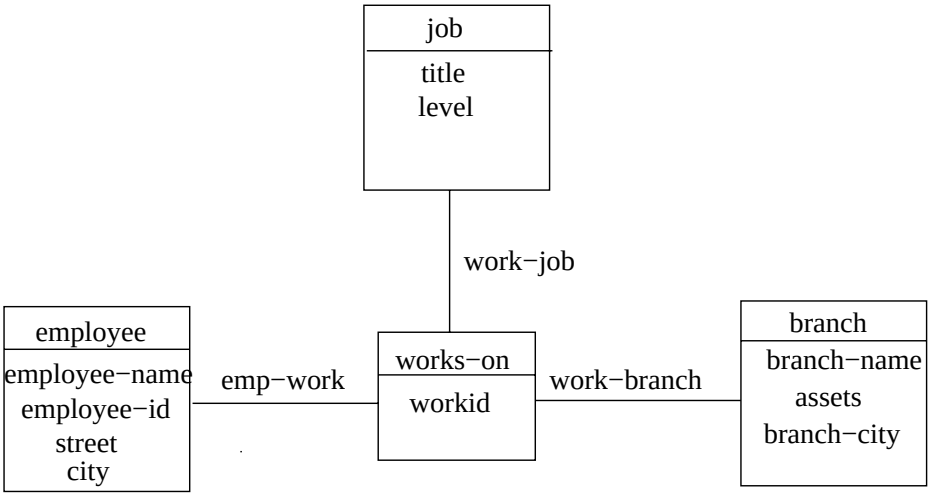


Figure 2.24 UML equivalent of Figure 2.13

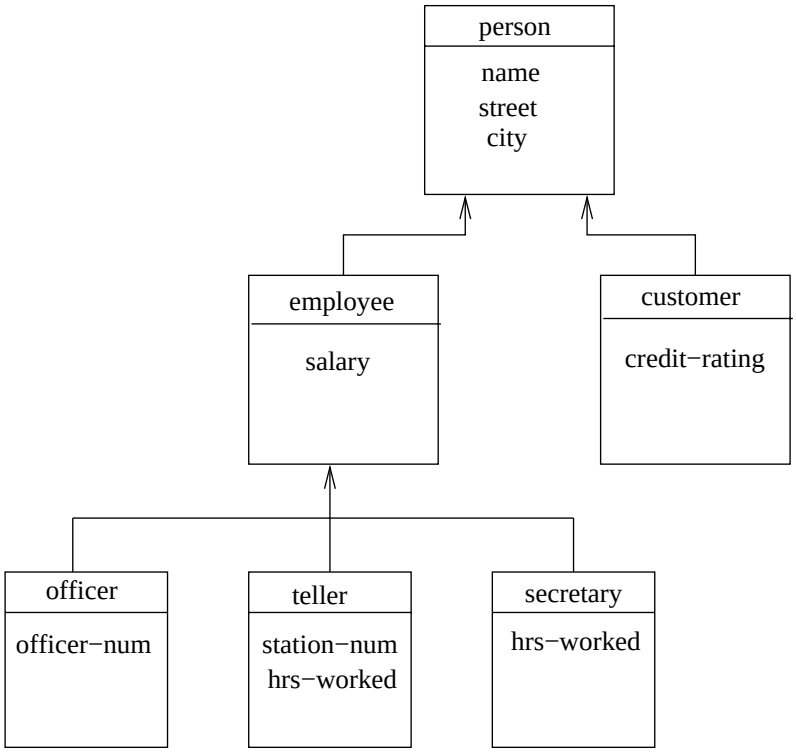


Figure 2.25 UML equivalent of Figure 2.17

Section 19.8.) If the merged bank is to have a single database, there are several potential problems:

- The possibility that the two original banks have branches with the same name
- The possibility that some customers are customers of both original banks
- The possibility that some loan or account numbers were used at both original banks (for different loans or accounts, of course)

For each of these potential problems, describe why there is indeed a potential for difficulties. Propose a solution to the problem. For your solution, explain any changes that would have to be made and describe what their effect would be on the schema and the data.

Answer: In this example, we assume that both banks have the shared identifiers for customers, such as the social security number. We see the general solution in the next exercise.

Each of the problems mentioned does have potential for difficulties.

a. *branch-name* is the primary-key of the *branch* entity set. Therefore while merging the two banks' entity sets, if both banks have a branch with the same name, one of them will be lost.

b. customers participate in the relationship sets *cust-banker*, *borrower* and *depositor*. While merging the two banks' *customer* entity sets, duplicate tuples of the same customer will be deleted. Therefore those relations in the three mentioned relationship sets which involved these deleted tuples will have to be updated. Note that if the tabular representation of a relationship set is obtained by taking a union of the primary keys of the participating entity sets, no modification to these relationship sets is required.

c. The problem caused by *loans* or *accounts* with the same number in both the banks is similar to the problem caused by branches in both the banks with the same *branch-name*.

To solve the problems caused by the merger, no schema changes are required. Merge the *customer* entity sets removing duplicate tuples with the same *social-security* field. Before merging the *branch* entity sets, prepend the old bank name to the *branch-name* attribute in each tuple. The *employee* entity sets can be merged directly, and so can the *payment* entity sets. No duplicate removal should be performed. Before merging the *loan* and *account* entity sets, whenever there is a number common in both the banks, the old number is replaced by a new unique number, in one of the banks.

Next the relationship sets can be merged. Any relation in any relationship set which involves a tuple which has been modified earlier due to the merger, is itself modified to retain the same meaning. For example let 1611 be a loan number common in both the banks prior to the merger, and let it be replaced by a new unique number 2611 in one of the banks, say bank 2. Now all the relations in *borrower*, *loan-branch* and *loan-payment* of bank 2 which refer to loan number 1611 will have to be modified to refer to 2611. Then the merger with bank 1's corresponding relationship sets can take place.

2.27 Reconsider the situation described for Exercise 2.26 under the assumption that one bank is in the United States and the other is in Canada. As before, the banks use the schema of Figure 2.22, except that the Canadian bank uses the *social-insurance* number assigned by the Canadian government, whereas the U.S. bank uses the social-security number to identify customers. What problems (beyond those identified in Exercise 2.24) might occur in this multinational case? How would you resolve them? Be sure to consider both the scheme and the actual data values in constructing your answer.

Answer: This is a case in which the schemas of the two banks differ, so the merger becomes more difficult. The identifying attribute for persons in the US is *social-security*, and in Canada it is *social-insurance*. Therefore the merged schema cannot use either of these. Instead we introduce a new attribute *person-id*, and use this uniformly for everybody in the merged schema. No other change to the schema is required. The values for the *person-id* attribute may be obtained by several ways. One way would be to prepend a country code to the old *social-security* or *social-insurance* values ("U" and "C" respectively, for instance), to get the corresponding *person-id* values. Another way would be to assign fresh numbers starting from 1 upwards, one number to each *social-security* and *social-insurance* value in the old databases.

Once this has been done, the actual merger can proceed as according to the answer to the previous question. If a particular relationship set, say *borrower*, involves only US customers, this can be expressed in the merged database by specializing the entity-set *customer* into *us-customer* and *canada-customer*, and making only *us-customer* participate in the merged *borrower*. Similarly *employee* can be specialized if needed.

CHAPTER 3

Relational Model

This chapter presents the relational model and three relational languages. The relational model (Section 3.1) is used extensively throughout the text as is the relational algebra (Section 3.2). The chapter also covers the tuple relational calculus (Section 3.6) and domain relational calculus (Section 3.7) (which is the basis of the QBE language described in Chapter 5). Classes that emphasize only SQL may omit the relational calculus languages.

Our notation for the tuple relational calculus makes it easy to present the concept of a safe query. The concept of safety for the domain relational calculus, though identical to that for the tuple calculus, is much more cumbersome notationally and requires careful presentation. This consideration may suggest placing somewhat less emphasis on the domain calculus for classes not planning to cover QBE.

Section 3.3 presents extended relational-algebra operations, such as outer-joins and aggregates. The evolution of query languages such as SQL clearly indicates the importance of such extended operations. Some of these operations, such as outer-joins can be expressed in the tuple/domain relational calculus, while extensions are required for other operations, such as aggregation. We have chosen not to present such extensions to the relational calculus, and instead restrict our attention to extensions of the algebra.

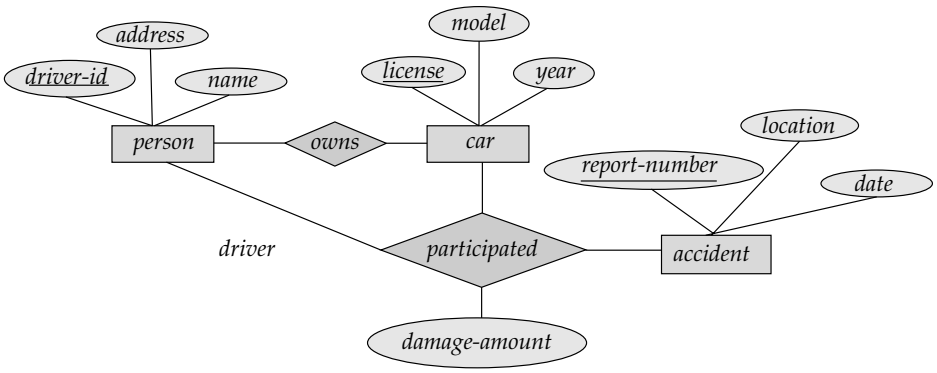


Figure 3.38. E-R diagram.

Exercises

3.1 Design a relational database for a university registrar's office. The office maintains data about each class, including the instructor, the number of students enrolled, and the time and place of the class meetings. For each student–class pair, a grade is recorded.
Answer: Underlined attributes indicate the primary key.

student (student-id, name, program)
course (courseno, title, syllabus, credits)
course-offering (courseno, secno, year, semester, time, room)
instructor (instructor-id, name, dept, title)
enrols (student-id, courseno, secno, semester, year, grade)
teaches (courseno, secno, semester, year, instructor-id)
requires (maincourse, prerequisite)

3.2 Describe the differences in meaning between the terms *relation* and *relation schema*. Illustrate your answer by referring to your solution to Exercise 3.1.
Answer: A relation schema is a type definition, and a relation is an instance of that schema. For example, *student* (*ss#*, *name*) is a relation schema and

ss#	name
123-45-6789	Tom Jones
456-78-9123	Joe Brown

is a relation based on that schema.

3.3 Design a relational database corresponding to the E-R diagram of Figure 3.38.
Answer: The relational database schema is given below.

person (driver-id, *name, address*)
car (license, *year*, *model*)
accident (report-number, *location*, *date*)
owns (driver-id, license)
participated (report-number *driver-id*, *license*, *damage-amount*)

employee (*person-name*, *street*, *city*)
works (*person-name*, *company-name*, *salary*)
company (*company-name*, *city*)
manages (*person-name*, *manager-name*)

Figure 3.39. Relational database for Exercises 3.5, 3.8 and 3.10.

3.4 In Chapter 2, we saw how to represent many-to-many, many-to-one, one-to-many, and one-to-one relationship sets. Explain how primary keys help us to represent such relationship sets in the relational model.
Answer: Suppose the primary key of relation schema R is $\{A_{i_1},\ A_{i_2}, ..., A_{i_n}\}$ and the primary key of relation schema S is $\{B_{i_1},\ B_{i_2}, ..., B_{i_m}\}$. Then a relationship between the 2 sets can be represented as a tuple $(A_{i_1},\ A_{i_2}, ..., A_{i_n}$ $B_{i_1},\ B_{i_2}, ..., B_{i_m})$. In a one-to-one relationship, each value on $\{A_{i_1},\ A_{i_2}, ..., A_{i_n}\}$ will appear in exactly one tuple and likewise for $\{B_{i_1},\ B_{i_2}, ..., B_{i_m}\}$. In a many-to-one relationship (e.g., many A - one B), each value on $\{A_{i_1},\ A_{i_2}, ..., A_{i_n}\}$ will appear once, and each value on $\{B_{i_1},\ B_{i_2}, ..., B_{i_n}\}$ may appear many times. In a many-to-many relationship, values on both $\{A_{i_1},\ A_{i_2}, ..., A_{i_n}\}$ and $\{B_{i_1}, B_{i_2}, ...,$ $B_{i_m}\}$ will appear many times. However, in all the above cases $\{A_{i_1},\ A_{i_2}, ..., A_{i_n}$, $B_{i_1},\ B_{i_2}, ..., B_{i_m}\}$ is a primary key, so no tuple on $(A_{j_1}, ..., A_{j_n}\ B_{k_1}, ..., B_{k_m})$ will appear more than once.

3.5 Consider the relational database of Figure 3.39, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:

- **a.** Find the names of all employees who work for First Bank Corporation.
- **b.** Find the names and cities of residence of all employees who work for First Bank Corporation.
- **c.** Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000 per annum.
- **d.** Find the names of all employees in this database who live in the same city as the company for which they work.
- **e.** Find the names of all employees who live in the same city and on the same street as do their managers.
- **f.** Find the names of all employees in this database who do not work for First Bank Corporation.
- **g.** Find the names of all employees who earn more than every employee of Small Bank Corporation.
- **h.** Assume the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

Answer:

- **a.** $\Pi_{person\text{-}name}\ (\sigma_{company\text{-}name} =$ "First Bank Corporation" $(works))$
- **b.** $\Pi_{person\text{-}name,\ city}\ (employee\ \bowtie$
  $(\sigma_{company\text{-}name} =$ "First Bank Corporation" $(works)))$

c. $\Pi_{person\text{-}name,\ street,\ city}$
$(\sigma_{(company\text{-}name\ =\ \text{“First Bank Corporation”}\ \wedge\ salary\ >\ 10000)}$
$works \bowtie employee)$

d. $\Pi_{person\text{-}name}\ (employee \bowtie works \bowtie company)$

e. $\Pi_{person\text{-}name}\ ((employee \bowtie manages)$
$\bowtie_{(manager\text{-}name\ =\ employee2.person\text{-}name\ \wedge\ employee.street\ =\ employee2.street}$
$\wedge\ employee.city\ =\ employee2.city)}(\rho_{employee2}\ (employee)))$

f. The following solutions assume that all people work for exactly one company. If one allows people to appear in the database (e.g. in *employee*) but not appear in *works*, the problem is more complicated. We give solutions for this more realistic case later.

$\Pi_{person\text{-}name}\ (\sigma_{company\text{-}name\ \neq\ \text{“First Bank Corporation”}}(works))$

If people may not work for any company:

$\Pi_{person\text{-}name}(employee)\ -\ \Pi_{person\text{-}name}$
$(\sigma_{(company\text{-}name\ =\ \text{“First Bank Corporation”})}(works))$

g. $\Pi_{person\text{-}name}\ (works)\ -\ (\Pi_{works.person\text{-}name}\ (works$
$\bowtie_{(works.salary\ \leq works2.salary\ \wedge\ works2.company\text{-}name\ =\ \text{“Small Bank Corporation”})}$
$\rho_{works2}(works)))$

h. Note: Small Bank Corporation will be included in each answer.

$\Pi_{company\text{-}name}\ (company \div$
$(\Pi_{city}\ (\sigma_{company\text{-}name\ =\ \text{“Small Bank Corporation”}}\ (company))))$

3.6 Consider the relation of Figure 3.21, which shows the result of the query “Find the names of all customers who have a loan at the bank.” Rewrite the query to include not only the name, but also the city of residence for each customer. Observe that now customer Jackson no longer appears in the result, even though Jackson does in fact have a loan from the bank.

a. Explain why Jackson does not appear in the result.

b. Suppose that you want Jackson to appear in the result. How would you modify the database to achieve this effect?

c. Again, suppose that you want Jackson to appear in the result. Write a query using an outer join that accomplishes this desire without your having to modify the database.

Answer: The rewritten query is

$\Pi_{customer\text{-}name,customer\text{-}city,amount}(borrower \bowtie loan \bowtie customer)$

a. Although Jackson does have a loan, no address is given for Jackson in the *customer* relation. Since no tuple in *customer* joins with the Jackson tuple of *borrower*, Jackson does not appear in the result.

b. The best solution is to insert Jackson’s address into the *customer* relation. If the address is unknown, null values may be used. If the database system does not support nulls, a special value may be used (such as **unknown**) for Jackson’s street and city. The special value chosen must not be a plausible name for an actual city or street.

c. $\Pi_{customer\text{-}name,customer\text{-}city,amount}((borrower \bowtie loan) \rhd\!\bowtie customer)$

3.7 The outer-join operations extend the natural-join operation so that tuples from the participating relations are not lost in the result of the join. Describe how the theta join operation can be extended so that tuples from the left, right, or both relations are not lost from the result of a theta join.

Answer:

a. The left outer theta join of *r*(*R*) and *s*(*S*) $(r \rhd\!\bowtie_\theta s)$ can be defined as
$(r \bowtie_\theta s) \cup ((r - \Pi_R(r \bowtie_\theta s)) \times (null, null, \ldots, null))$
The tuple of nulls is of size equal to the number of attributes in *S*.

b. The right outer theta join of *r*(*R*) and *s*(*S*) $(r \bowtie\!\lhd_\theta s)$ can be defined as
$(r \bowtie_\theta s) \cup ((null, null, \ldots, null) \times (s - \Pi_S(r \bowtie_\theta s)))$
The tuple of nulls is of size equal to the number of attributes in *R*.

c. The full outer theta join of *r*(*R*) and *s*(*S*) $(r \rhd\!\bowtie\!\lhd_\theta s)$ can be defined as
$(r \bowtie_\theta s) \cup ((null, null, \ldots, null) \times (s - \Pi_S(r \bowtie_\theta s))) \cup$
$((r - \Pi_R(r \bowtie_\theta s)) \times (null, null, \ldots, null))$
The first tuple of nulls is of size equal to the number of attributes in *R*, and the second one is of size equal to the number of attributes in *S*.

3.8 Consider the relational database of Figure 3.39. Give an expression in the relational algebra for each request:

a. Modify the database so that Jones now lives in Newtown.

b. Give all employees of First Bank Corporation a 10 percent salary raise.

c. Give all managers in this database a 10 percent salary raise.

d. Give all managers in this database a 10 percent salary raise, unless the salary would be greater than $100,000. In such cases, give only a 3 percent raise.

e. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

Answer:

a. $employee \leftarrow \Pi_{person\text{-}name,street,\text{“}Newtown\text{”}}$
$(\sigma_{person\text{-}name=\text{“Jones”}}(employee))$
$\cup (employee - \sigma_{person\text{-}name=\text{“Jones”}}(employee))$

b. $works \leftarrow \Pi_{person\text{-}name,company\text{-}name,1.1*salary}($
$\sigma_{(company\text{-}name=\text{“First Bank Corporation”})}(works))$
$\cup (works - \sigma_{company\text{-}name=\text{“First Bank Corporation”}}(works))$

c. The update syntax allows reference to a single relation only. Since this update requires access to both the relation to be updated (*works*) and the *manages* relation, we must use several steps. First we identify the tuples of *works* to be updated and store them in a temporary relation (t_1). Then we create a temporary relation containing the new tuples (t_2). Finally, we delete the tuples in t_1, from *works* and insert the tuples of t_2.

$t_1 \leftarrow \Pi_{works.person\text{-}name,company\text{-}name,salary}$
$(\sigma_{works.person\text{-}name=manager\text{-}name}(works \times manages))$

$t_2 \leftarrow \Pi_{person\text{-}name,company\text{-}name,1.1*salary}(t_1)$

$works \leftarrow (works - t_1) \cup t_2$

d. The same situation arises here. As before, t_1, holds the tuples to be updated and t_2 holds these tuples in their updated form.

$t_1 \leftarrow \Pi_{works.person\text{-}name,company\text{-}name,salary}$
$(\sigma_{works.person\text{-}name=manager\text{-}name}(works \times manages))$

$t_2 \leftarrow \Pi_{works.person\text{-}name,company\text{-}name,salary*1.03}$
$(\sigma_{t_1.salary\ *\ 1.1\ >\ 100000}(t_1))$

$t_2 \leftarrow t_2 \cup (\Pi_{works.person\text{-}name,company\text{-}name,salary*1.1}$
$(\sigma_{t_1.salary\ *\ 1.1\ \leq\ 100000}(t_1)))$

$works \leftarrow (works - t_1) \cup t_2$

e. $works \leftarrow works - \sigma_{company-name=}$"Small Bank Corporation"$(works)$

3.9 Using the bank example, write relational-algebra queries to find the accounts held by more than two customers in the following ways:

a. Using an aggregate function.

b. Without using any aggregate functions.

Answer:

a. $t_1 \leftarrow {}_{account\text{-}number}\mathcal{G}_{\textbf{count}\ customer\text{-}name}(depositor)$
$\Pi_{account\text{-}number}\ (\sigma_{num\text{-}holders>2}\ (\rho_{account\text{-}holders(account\text{-}number,num\text{-}holders)}(t_1)))$

b. $t_1 \leftarrow (\rho_{d1}(depositor) \times \rho_{d2}(depositor) \times \rho_{d3}(depositor))$
$t_2 \leftarrow \sigma_{(d1.account\text{-}number=d2.account\text{-}number=d3.account\text{-}number)}(t_1)$
$\Pi_{d1.account\text{-}number}(\sigma_{(d1.customer\text{-}name\neq d2.customer\text{-}name\ \wedge}$
$_{d2.customer\text{-}name\neq d3.customer\text{-}name\ \wedge d3.customer\text{-}name\neq d1.customer\text{-}name)}(t_2))$

3.10 Consider the relational database of Figure 3.39. Give a relational-algebra expression for each of the following queries:

a. Find the company with the most employees.

b. Find the company with the smallest payroll.

c. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

Answer:

a. $t_1 \leftarrow {}_{company\text{-}name}\mathcal{G}_{\textbf{count-distinct}\ person\text{-}name}(works)$
$t_2 \leftarrow \textbf{max}_{num\text{-}employees}(\rho_{company\text{-}strength(company\text{-}name,num\text{-}employees)}(t_1))$
$\Pi_{company\text{-}name}(\rho_{t_3(company\text{-}name,num\text{-}employees)}(t_1) \bowtie \rho_{t_4(num\text{-}employees)}(t_2))$

b. $t_1 \leftarrow {}_{company\text{-}name}\mathcal{G}_{\textbf{sum}\ salary}(works)$
$t_2 \leftarrow \textbf{min}_{payroll}(\rho_{company\text{-}payroll(company\text{-}name,payroll)}(t_1))$
$\Pi_{company\text{-}name}(\rho_{t_3(company\text{-}name,payroll)}(t_1) \bowtie \rho_{t_4(payroll)}(t_2))$

c. $t_1 \leftarrow {}_{company\text{-}name}\mathcal{G}_{\textbf{avg}\ salary}(works)$
$t_2 \leftarrow \sigma_{company\text{-}name\ =}$ "First Bank Corporation"(t_1)

$$\Pi_{t_3.company\text{-}name}((\rho_{t_3(company\text{-}name,avg\text{-}salary)}(t_1)) \bowtie_{t_3.avg\text{-}salary > first\text{-}bank.avg\text{-}salary} (\rho_{first\text{-}bank(company\text{-}name,avg\text{-}salary)}(t_2)))$$

3.11 List two reasons why we may choose to define a view.
Answer:

a. Security conditions may require that the entire logical database be not visible to all users.
b. We may wish to create a personalized collection of relations that is better matched to a certain user's intuition than is the actual logical model.

3.12 List two major problems with processing update operations expressed in terms of views.
Answer: Views present significant problems if updates are expressed with them. The difficulty is that a modification to the database expressed in terms of a view must be translated to a modification to the actual relations in the logical model of the database.

a. Since the view may not have all the attributes of the underlying tables, insertion of a tuple into the view will insert tuples into the underlying tables, with those attributes not participating in the view getting null values. This may not be desirable, especially if the attribute in question is part of the primary key of the table.
b. If a view is a join of several underlying tables and an insertion results in tuples with nulls in the join columns, the desired effect of the insertion will not be achieved. In other words, an update to a view may not be expressible at all as updates to base relations. For an explanatory example, see the *loan-info* updation example in Section 3.5.2.

3.13 Let the following relation schemas be given:

$$\begin{aligned} R &= (A, B, C) \\ S &= (D, E, F) \end{aligned}$$

Let relations $r(R)$ and $s(S)$ be given. Give an expression in the tuple relational calculus that is equivalent to each of the following:

a. $\Pi_A(r)$
b. $\sigma_{B=17}(r)$
c. $r \times s$
d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

Answer:

a. $\{t \mid \exists\, q \in r\, (q[A] = t[A])\}$
b. $\{t \mid t \in r \wedge t[B] = 17\}$
c. $\{t \mid \exists\, p \in r\, \exists\, q \in s\, (t[A] = p[A] \wedge t[B] = p[B] \wedge t[C] = p[C] \wedge t[D] = q[D] \wedge t[E] = q[E] \wedge t[F] = q[F])\}$
d. $\{t \mid \exists\, p \in r\, \exists\, q \in s\, (t[A] = p[A] \wedge t[F] = q[F] \wedge p[C] = q[D]\}$

3.14 Let $R = (A, B, C)$, and let r_1 and r_2 both be relations on schema R. Give an expression in the domain relational calculus that is equivalent to each of the following:

a. $\Pi_A(r_1)$
b. $\sigma_{B=17}(r_1)$
c. $r_1 \cup r_2$
d. $r_1 \cap r_2$
e. $r_1 - r_2$
f. $\Pi_{A,B}(r_1) \bowtie \Pi_{B,C}(r_2)$

Answer:

a. $\{< t > \mid \exists\, p,\ q\ (< t, p, q > \in\ r_1)\}$
b. $\{< a, b, c > \mid < a, b, c > \in\ r_1 \wedge\ b = 17\}$
c. $\{< a, b, c > \mid < a, b, c > \in\ r_1 \vee\ < a, b, c > \in\ r_2\}$
d. $\{< a, b, c > \mid < a, b, c > \in\ r_1 \wedge\ < a, b, c > \in\ r_2\}$
e. $\{< a, b, c > \mid < a, b, c > \in\ r_1 \wedge\ < a, b, c > \notin\ r_2\}$
f. $\{< a, b, c > \mid \exists\, p,\ q\ (< a, b, p > \in\ r_1 \wedge\ < q, b, c > \in\ r_2)\}$

3.15 Repeat Exercise 3.5 using the tuple relational calculus and the domain relational calculus.
Answer:

a. Find the names of all employees who work for First Bank Corporation:-
  i. $\{t \mid \exists\, s \in\ works\ (t[person\text{-}name] = s[person\text{-}name] \wedge s[company\text{-}name] =$ "First Bank Corporation"$)\}$
  ii. $\{< p > \mid \exists\, c,\ s\ (< p, c, s > \in\ works \wedge c =$ "First Bank Corporation"$)\}$

b. Find the names and cities of residence of all employees who work for First Bank Corporation:-
  i. $\{t \mid \exists\, r \in\ employee\ \exists\, s \in\ works\ (\ t[person\text{-}name] = r[person\text{-}name] \wedge t[city] = r[city] \wedge r[person\text{-}name] = s[person\text{-}name] \wedge s[company\text{-}name] =$ "First Bank Corporation"$)\}$
  ii. $\{< p, c > \mid \exists\, co,\ sa,\ st\ (< p, co, sa > \in\ works \wedge\ < p, st, c > \in\ employee \wedge\ co =$ "First Bank Corporation"$)\}$

c. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000 per annum:-
  i. $\{t \mid t \in\ employee \wedge (\exists\, s \in\ works\ (\ s[person\text{-}name] = t[person\text{-}name] \wedge s[company\text{-}name] =$ "First Bank Corporation" $\wedge\ s[salary] > 10000))\}$
  ii. $\{< p, s, c > \mid < p, s, c > \in\ employee \wedge\ \exists\, co,\ sa\ (< p, co, sa > \in\ works \wedge co =$ "First Bank Corporation" $\wedge\ sa > 10000)\}$

d. Find the names of all employees in this database who live in the same city as the company for which they work:-
  i. $\{t \mid \exists\, e \in\ employee\ \exists\, w \in\ works\ \exists\, c \in\ company\ (t[person\text{-}name] = e[person\text{-}name] \wedge e[person\text{-}name] = w[person\text{-}name] \wedge w[company\text{-}name] = c[company\text{-}name] \wedge e[city] = c[city])\}$

ii. $\{< p > \mid \exists\, st,\ c,\ co,\ sa\ (< p, st, c > \in\ employee$
$\wedge\ < p, co, sa > \in\ works \wedge\ < co, c > \in\ company)\}$

e. Find the names of all employees who live in the same city and on the same street as do their managers:-

i. $\{\ t \mid \exists\, l\ \in\ employee\, \exists\, m\ \in\ manages\, \exists\, r\ \in\ employee$
$(l[person\text{-}name] = m[person\text{-}name] \wedge m[manager\text{-}name] = r[person\text{-}name]$
$\wedge\ l[street] = r[street] \wedge l[city] = r[city] \wedge t[person\text{-}name] = l[person\text{-}name])\}$

ii. $\{< t > \mid \exists\, s,\ c,\ m\ (< t, s, c > \in\ employee \wedge\ < t, m > \in\ manages \wedge\ < m, s, c > \in\ employee)\}$

f. Find the names of all employees in this database who do not work for First Bank Corporation:-

If one allows people to appear in the database (e.g. in *employee*) but not appear in *works*, the problem is more complicated. We give solutions for this more realistic case later.

i. $\{\ t \mid \exists\, w\ \in\ works\ (\ w[company\text{-}name] \neq$ "First Bank Corporation"
$\wedge\, t[person\text{-}name] = w[person\text{-}name])\}$

ii. $\{< p > \mid \exists\, c,\ s\ (< p, c, s > \in\ works \wedge\ c \neq$ "First Bank Corporation")$\}$

If people may not work for any company:

i. $\{\ t \mid \exists\, e\ \in\ employee\ (\ t[person\text{-}name] = e[person\text{-}name] \wedge \neg\, \exists\, w\ \in\ works$
$(w[company\text{-}name] =$ "First Bank Corporation"
$\wedge\, w[person\text{-}name] = t[person\text{-}name]))\}$

ii. $\{\ < p > \mid \exists\, s,\ c\ (< p, s, c > \in\ employee) \wedge\ \neg\, \exists\, x,\ y$
$(y =$ "First Bank Corporation"$\wedge\ < p, y, x > \in\ works)\}$

g. Find the names of all employees who earn more than every employee of Small Bank Corporation:-

i. $\{t \mid \exists\, w\ \in\ works\ (t[person\text{-}name] = w[person\text{-}name] \wedge\ \forall\, s\ \in\ works$
$(s[company\text{-}name] =$ "Small Bank Corporation" $\Rightarrow\ w[salary] > s[salary]))\}$

ii. $\{< p > \mid \exists\, c,\ s\ (< p, c, s > \in\ works\ \wedge\ \forall\, p_2,\ c_2,\ s_2$
$(< p_2, c_2, s_2 > \notin\ works \vee\ c_2 \neq$ "Small Bank Corporation" $\vee\ s > s_2))\}$

h. Assume the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

Note: Small Bank Corporation will be included in each answer.

i. $\{t \mid \forall\, s\ \in\ company\ (s[company\text{-}name] =$ "Small Bank Corporation" $\Rightarrow$
$\exists\, r\ \in\ company\, (t[company\text{-}name] = r[company\text{-}name] \wedge r[city] = s[city]))\}$

ii. $\{< co > \mid \forall\, co_2,\ ci_2\ (< co_2, ci_2 > \notin\ company$
$\vee\ co_2 \neq$ "Small Bank Corporation" $\vee\ < co, ci_2 > \in\ company)\}$

3.16 Let $R = (A,\ B)$ and $S = (A,\ C)$, and let $r(R)$ and $s(S)$ be relations. Write relational-algebra expressions equivalent to the following domain-relational-calculus expressions:

a. $\{< a > \mid \exists\, b\, (< a, b > \in r \wedge b = 17)\}$
b. $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s\}$
c. $\{< a > \mid \exists\, b\, (< a, b > \in r) \vee \forall\, c\, (\exists\, d\, (< d, c > \in s) \Rightarrow < a, c > \in s)\}$
d. $\{< a > \mid \exists\, c\, (< a, c > \in s \wedge \exists\, b_1, b_2\, (< a, b_1 > \in r \wedge < c, b_2 > \in r \wedge b_1 > b_2))\}$

Answer:

a. $\Pi_A\, (\sigma_{B\,=\,17}\, (r))$
b. $r \bowtie s$
c. $\Pi_A(r) \cup (r \div \sigma_B(\Pi_C(s)))$
d. $\Pi_{r.A}\, ((r \bowtie s) \bowtie_{c\,=\,r2.A\,\wedge\,r.B\,>\,r2.B} (\rho_{r_2}(r)))$

It is interesting to note that (*d*) is an abstraction of the notorious query "Find all employees who earn more than their manager." Let $R = (emp,\ sal)$, $S = (emp,\ mgr)$ to observe this.

3.17 Let $R = (A,\ B)$ and $S = (A,\ C)$, and let $r(R)$ and $s(S)$ be relations. Using the special constant *null*, write tuple-relational-calculus expressions equivalent to each of the following:

a. $r \bowtie\!\!\!\!\sqsubset s$
b. $r \sqsupset\!\!\!\!\bowtie\!\!\!\!\sqsubset s$
c. $r \sqsupset\!\!\!\!\bowtie s$

Answer:

a. $\{t \mid \exists r \in R\, \exists s \in S\, (r[A] = s[A] \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = s[C]) \vee \exists s \in S(\neg \exists r \in R(r[A] = s[A]) \wedge t[A] = s[A] \wedge t[C] = s[C] \wedge t[B] = null)\}$
b. $\{t \mid \exists r \in R\, \exists s \in S\, (r[A] = s[A] \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = s[C]) \vee \exists r \in R(\neg \exists s \in S(r[A] = s[A]) \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = null) \vee \exists s \in S(\neg \exists r \in R(r[A] = s[A]) \wedge t[A] = s[A] \wedge t[C] = s[C] \wedge t[B] = null)\}$
c. $\{t \mid \exists r \in R\, \exists s \in S\, (r[A] = s[A] \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = s[C]) \vee \exists r \in R(\neg \exists s \in S(r[A] = s[A]) \wedge t[A] = r[A] \wedge t[B] = r[B] \wedge t[C] = null)\}$

3.18 List two reasons why null values might be introduced into the database.
Answer: Nulls may be introduced into the database because the actual value is either unknown or does not exist. For example, an employee whose address has changed and whose new address is not yet known should be retained with a null address. If employee tuples have a composite attribute *dependents*, and a particular employee has no dependents, then that tuple's *dependents* attribute should be given a null value.

3.19 Certain systems allow *marked* nulls. A marked null $\perp_i$ is equal to itself, but if $i \neq j$, then $\perp_i \neq \perp_j$. One application of marked nulls is to allow certain updates through views. Consider the view *loan-info* (Section 3.5). Show how you can use marked nulls to allow the insertion of the tuple ("Johnson", 1900) through *loan-info*.
Answer: To insert the tuple ("Johnson", 1900) into the view *loan-info*, we can do

the following:-

$borrower \leftarrow (\text{“Johnson”}, \perp_k) \cup borrower$

$loan \leftarrow (\perp_k, \perp, 1900) \cup loan$

such that $\perp_k$ is a new marked null not already existing in the database.

CHAPTER 4

SQL

Chapter 4 covers the relational language SQL. The discussion is based on SQL-92, since the more recent SQL:1999 is not widely supported yet. Extensions provided by SQL:1999 are covered later in Chapters 9 and 22. Integrity constraint and authorization features of SQL-92 are described in Chapter 6. SQL being a large language, many of its features are not covered here, and are not appropriate for an introductory course on databases. Standard books on SQL, such as Date and Darwen [1993] and Melton and Simon [1993], or the system manuals of the database system you use can be used as supplements for students who want to delve deeper into the intricacies of SQL.

Although it is possible to cover this chapter using only handwritten exercises, we strongly recommend providing access to an actual database system that supports SQL. A style of exercise we have used is to create a moderately large database and give students a list of queries in English to write and run using SQL. We publish the actual answers (that is the result relations they should get, not the SQL they must enter). By using a moderately large database, the probability that a "wrong" SQL query will just happen to return the "right" result relation can be made very small. This approach allows students to check their own answers for correctness immediately rather than wait for grading and thereby it speeds up the learning process. A few such example databases are available on the Web home page of this book.

Exercises that pertain to database design are best deferred until after Chapter 7.

Given the fact that the ODBC and JDBC protocols are fast becoming a primary means of accessing databases, we have significantly extended our coverage of these two protocols, including some examples. However, our coverage is only introductory, and omits many details that are useful in practise. Online tutorials/manuals or textbooks covering these protocols should be used as supplements, to help students make full use of the protocols.

Changes from 3rd edition:
Our coverage of SQL has been expanded to include the **with** clause, ODBC, JDBC, and schemas, catalogs and environments (Section 4.14).

Exercises

4.1 Consider the insurance database of Figure 4.12, where the primary keys are underlined. Construct the following SQL queries for this relational database.

a. Find the total number of people who owned cars that were involved in accidents in 1989.
b. Find the number of accidents in which the cars belonging to "John Smith" were involved.
c. Add a new accident to the database; assume any values for required attributes.
d. Delete the Mazda belonging to "John Smith".
e. Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

Answer: Note: The *participated* relation relates drivers, cars, and accidents.

a. Find the total number of people who owned cars that were involved in accidents in 1989.

Note: this is not the same as the total number of accidents in 1989. We must count people with several accidents only once.

```
select   count (distinct name)
from     accident, participated, person
where    accident.report-number = participated.report-number
and      participated.driver-id = person.driver-id
and      date between date '1989-00-00' and date '1989-12-31'
```

b. Find the number of accidents in which the cars belonging to "John Smith" were involved.

```
select   count (distinct *)
from     accident
where    exists
         (select *
          from participated, person
          where participated.driver-id = person.driver-id
                and person.name = 'John Smith'
                and accident.report-number = participated.report-number)
```

c. Add a new accident to the database; assume any values for required attributes.

We assume the driver was "Jones," although it could be someone else. Also, we assume "Jones" owns one Toyota. First we must find the license of the given car. Then the *participated* and *accident* relations must be updated in order to both record the accident and tie it to the given car. We assume values "Berkeley" for *location*, '2001-09-01' for date and *date*, 4007 for *report-number* and 3000 for damage amount.

person (*driver-id*, *name*, *address*)
car (*license*, *model*, *year*)
accident (*report-number*, *date*, *location*)
owns (*driver-id*, *license*)
participated (*driver-id*, *car*, *report-number*, *damage-amount*)

Figure 4.12. Insurance database.

```
insert into accident
        values (4007, '2001-09-01', 'Berkeley')

insert into participated
        select o.driver-id, c.license, 4007, 3000
        from person p, owns o, car c
        where p.name = 'Jones' and p.driver-id = o.driver-id and
                o.license = c.license and c.model = 'Toyota'
```

d. Delete the Mazda belonging to "John Smith".

Since *model* is not a key of the *car* relation, we can either assume that only one of John Smith's cars is a Mazda, or delete all of John Smith's Mazdas (the query is the same). Again assume *name* is a key for *person*.

```
delete car
where model = 'Mazda' and license in
    (select license
     from person p, owns o
     where p.name = 'John Smith' and p.driver-id = o.driver-id)
```

Note: The *owns*, *accident* and *participated* records associated with the Mazda still exist.

e. Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

```
update participated
set damage-amount = 3000
where report-number = "AR2197" and driver-id in
      (select driver-id
       from owns
       where license = "AABB2000")
```

4.2 Consider the employee database of Figure 4.13, where the primary keys are underlined. Give an expression in SQL for each of the following queries.

a. Find the names of all employees who work for First Bank Corporation.
b. Find the names and cities of residence of all employees who work for First Bank Corporation.
c. Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

d. Find all employees in the database who live in the same cities as the companies for which they work.
e. Find all employees in the database who live in the same cities and on the same streets as do their managers.
f. Find all employees in the database who do not work for First Bank Corporation.
g. Find all employees in the database who earn more than each employee of Small Bank Corporation.
h. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.
i. Find all employees who earn more than the average salary of all employees of their company.
j. Find the company that has the most employees.
k. Find the company that has the smallest payroll.
l. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

Answer:

a. Find the names of all employees who work for First Bank Corporation.

```
select employee-name
from works
where company-name = 'First Bank Corporation'
```

b. Find the names and cities of residence of all employees who work for First Bank Corporation.

```
select e.employee-name, city
from employee e, works w
where w.company-name = 'First Bank Corporation' and
      w.employee-name = e.employee-name
```

c. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000.

If people may work for several companies, the following solution will only list those who earn more than $10,000 per annum from "First Bank Corporation" alone.

```
select *
from employee
where employee-name in
    (select employee-name
     from works
     where company-name = 'First Bank Corporation' and salary ¿ 10000)
```

As in the solution to the previous query, we can use a join to solve this one also.

d. Find all employees in the database who live in the same cities as the companies for which they work.

```
select e.employee-name
from employee e, works w, company c
where e.employee-name = w.employee-name and e.city = c.city and
      w.company -name = c.company -name
```

e. Find all employees in the database who live in the same cities and on the same streets as do their managers.

```
select P.employee-name
from employee P, employee R, manages M
where P.employee-name = M.employee-name and
      M.manager-name = R.employee-name and
      P.street = R.street and P.city = R.city
```

f. Find all employees in the database who do not work for First Bank Corporation.
The following solution assumes that all people work for exactly one company.

```
select employee-name
from works
where company-name ≠ 'First Bank Corporation'
```

If one allows people to appear in the database (e.g. in *employee*) but not appear in *works*, or if people may have jobs with more than one company, the solution is slightly more complicated.

```
select employee-name
from employee
where employee-name not in
    (select employee-name
     from works
     where company-name = 'First Bank Corporation')
```

g. Find all employees in the database who earn more than every employee of Small Bank Corporation.

The following solution assumes that all people work for at most one company.

```
select employee-name
from works
where salary > all
    (select salary
     from works
     where company-name = 'Small Bank Corporation')
```

If people may work for several companies and we wish to consider the *total* earnings of each person, the problem is more complex. It can be solved by using a nested subquery, but we illustrate below how to solve it using the **with** clause.

```
with emp-total-salary as
   (select employee-name, sum(salary) as total-salary
    from works
    group by employee-name
   )
select employee-name
from emp-total-salary
where total-salary > all
   (select total-salary
    from emp-total-salary, works
    where works.company-name = 'Small Bank Corporation' and
          emp-total-salary.employee-name = works.employee-name
   )
```

h. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

The simplest solution uses the **contains** comparison which was included in the original System R Sequel language but is not present in the subsequent SQL versions.

```
select T.company-name
from company T
where (select R.city
       from company R
       where R.company-name = T.company-name)
     contains
       (select S.city
        from company S
        where S.company-name = 'Small Bank Corporation')
```

Below is a solution using standard SQL.

```
select S.company-name
from company S
where not exists ((select city
                   from company
                   where company-name = 'Small Bank Corporation')
                  except
                   (select city
                    from company T
                    where S.company-name = T.company-name))
```

i. Find all employees who earn more than the average salary of all employees of their company.

The following solution assumes that all people work for at most one company.

employee (*employee-name*, *street*, *city*)
works (*employee-name*, *company-name*, *salary*)
company (*company-name*, *city*)
manages (*employee-name*, *manager-name*)

Figure 4.13. Employee database.

```
select employee-name
from works T
where salary > (select avg (salary)
                from works S
                where T.company-name = S.company-name)
```

j. Find the company that has the most employees.

```
select company-name
from works
group by company-name
having count (distinct employee-name) >= all
    (select count (distinct employee-name)
     from works
     group by company-name)
```

k. Find the company that has the smallest payroll.

```
select company-name
from works
group by company-name
having sum (salary) <= all (select sum (salary)
                            from works
                            group by company-name)
```

l. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

```
select company-name
from works
group by company-name
having avg (salary) > (select avg (salary)
                       from works
                       where company-name = 'First Bank Corporation')
```

4.3 Consider the relational database of Figure 4.13. Give an expression in SQL for each of the following queries.

a. Modify the database so that Jones now lives in Newtown.
b. Give all employees of First Bank Corporation a 10 percent raise.
c. Give all managers of First Bank Corporation a 10 percent raise.
d. Give all managers of First Bank Corporation a 10 percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3 percent raise.

e. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

Answer: The solution for part 0.a assumes that each person has only one tuple in the *employee* relation. The solutions to parts 0.c and 0.d assume that each person works for at most one company.

a. Modify the database so that Jones now lives in Newtown.

```
update employee
set city = 'Newton'
where person-name = 'Jones'
```

b. Give all employees of First Bank Corporation a 10-percent raise.

```
update works
set salary = salary * 1.1
where company-name = 'First Bank Corporation'
```

c. Give all managers of First Bank Corporation a 10-percent raise.

```
update works
set salary = salary * 1.1
where employee-name in (select manager-name
                        from manages)
      and company-name = 'First Bank Corporation'
```

d. Give all managers of First Bank Corporation a 10-percent raise unless the salary becomes greater than $100,000; in such cases, give only a 3-percent raise.

```
update works T
set T.salary = T.salary * 1.03
where T.employee-name in (select manager-name
                          from manages)
      and T.salary * 1.1 > 100000
      and T.company-name = 'First Bank Corporation'

update works T
set T.salary = T.salary * 1.1
where T.employee-name in (select manager-name
                          from manages)
      and T.salary * 1.1 <= 100000
      and T.company-name = 'First Bank Corporation'
```

SQL-92 provides a **case** operation (see Exercise 4.11), using which we give a more concise solution:-

```
update works T
set T.salary = T.salary *
    (case
        when (T.salary * 1.1 > 100000) then 1.03
        else 1.1
    )
where T.employee-name in (select manager-name
                          from manages) and
      T.company-name = 'First Bank Corporation'
```

e. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

```
delete works
where company-name = 'Small Bank Corporation'
```

4.4 Let the following relation schemas be given:

$$R = (A, B, C)$$
$$S = (D, E, F)$$

Let relations $r(R)$ and $s(S)$ be given. Give an expression in SQL that is equivalent to each of the following queries.

a. $\Pi_A(r)$
b. $\sigma_{B=17}(r)$
c. $r \times s$
d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

Answer:

a. $\Pi_A(r)$

```
select distinct A
from r
```

b. $\sigma_{B=17}(r)$

```
select *
from r
where B = 17
```

c. $r \times s$

```
select distinct *
from r, s
```

d. $\Pi_{A,F}(\sigma_{C=D}(r \times s))$

```
select distinct A, F
from r, s
where C = D
```

4.5 Let $R = (A, B, C)$, and let r_1 and r_2 both be relations on schema R. Give an expression in SQL that is equivalent to each of the following queries.

a. $r_1 \cup r_2$
b. $r_1 \cap r_2$

c. $r_1 - r_2$
d. $\Pi_{AB}(r_1) \bowtie \Pi_{BC}(r_2)$

Answer:

a. $r_1 \cup r_2$

```
(select *
 from r1)
union
(select *
 from r2)
```

b. $r_1 \cap r_2$

We can write this using the **intersect** operation, which is the preferred approach, but for variety we present an solution using a nested subquery.

```
select *
from r1
where (A, B, C) in (select *
                    from r2)
```

c. $r_1 - r_2$

```
select *
from r1
where (A, B, C) not in (select *
                        from r2)
```

This can also be solved using the **except** clause.

d. $\Pi_{AB}(r_1) \bowtie \Pi_{BC}(r_2)$

```
select r1.A, r2.B, r3.C
from r1, r2
where r1.B = r2.B
```

4.6 Let $R = (A, B)$ and $S = (A, C)$, and let $r(R)$ and $s(S)$ be relations. Write an expression in SQL for each of the queries below:

a. $\{< a > \mid \exists\, b\, (< a, b > \in r \wedge b = 17)\}$
b. $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s\}$
c. $\{< a > \mid \exists\, c\, (< a, c > \in s \wedge \exists\, b_1, b_2\, (< a, b_1 > \in r \wedge < c, b_2 > \in r \wedge b_1 > b_2))\}$

Answer:

a. $\{< a > \mid \exists\, b\, (< a, b > \in r \wedge b = 17)\}$

```
select distinct A
from r
where B = 17
```

b. $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s)\}$

select distinct $r.A$, $r.B$, $s.C$
from r, s
where $r.A = s.A$

c. $\{< a > \mid \exists\, c\, (< a, c > \in s \wedge \exists\, b_1, b_2\, (< a, b_1 > \in r \wedge < c, b_2 > \in r \wedge b_1 > b_2))\}$

select distinct $s.A$
from $s,\ r\ e,\ r\ m$
where $s.A = e.A$ **and** $s.C = m.A$ **and** $e.B > m.B$

4.7 Show that, in SQL, $<>$ **all** is identical to **not in**.
Answer: Let the set S denote the result of an SQL subquery. We compare $(x <>$ **all** $S)$ with $(x$ **not in** $S)$. If a particular value x_1 satisfies $(x_1 <>$ **all** $S)$ then for all elements y of $S\ x_1 \neq y$. Thus x_1 is not a member of S and must satisfy $(x_1$ **not in** $S)$. Similarly, suppose there is a particular value x_2 which satisfies $(x_2$ **not in** $S)$. It cannot be equal to any element w belonging to S, and hence $(x_2 <>$ **all** $S)$ will be satisfied. Therefore the two expressions are equivalent.

4.8 Consider the relational database of Figure 4.13. Using SQL, define a view consisting of *manager-name* and the average salary of all employees who work for that manager. Explain why the database system should not allow updates to be expressed in terms of this view.
Answer:

create view *salinfo* **as**
select *manager-name*, **avg**(*salary*)
from *manages m*, *works w*
where *m.employee-name* = *w.employee-name*
group by *manager-name*

Updates should not be allowed in this view because there is no way to determine how to change the underlying data. For example, suppose the request is "change the average salary of employees working for Smith to $200". Should everybody who works for Smith have their salary changed to $200? Or should the first (or more, if necessary) employee found who works for Smith have their salary adjusted so that the average is $200? Neither approach really makes sense.

4.9 Consider the SQL query

select *p.a*1
from *p*, *r*1, *r*2
where *p.a*1 = *r*1.*a*1 **or** *p.a*1 = *r*2.*a*1

Under what conditions does the preceding query select values of $p.a1$ that are either in $r1$ or in $r2$? Examine carefully the cases where one of $r1$ or $r2$ may be empty.
Answer: The query selects those values of *p.a1* that are equal to some value of *r1.a1* or *r2.a1* if and only if both *r1* and *r2* are non-empty. If one or both of *r1* and

r2 are empty, the cartesian product of *p, r1* and *r2* is empty, hence the result of the query is empty. Of course if *p* itself is empty, the result is as expected, i.e. empty.

4.10 Write an SQL query, without using a **with** clause, to find all branches where the total account deposit is less than the average total account deposit at all branches,

a. Using a nested query in the **from** clauser.
b. Using a nested query in a **having** clause.

Answer: We output the branch names along with the total account deposit at the branch.

a. Using a nested query in the **from** clauser.

```
select branch-name, tot-balance
from (select branch-name, sum (balance)
      from account
      group by branch-name) as branch-total(branch-name, tot-balance)
where tot-balance ¡
      ( select avg (tot-balance)
      from ( select branch-name, sum (balance)
            from account
            group by branch-name) as branch-total(branch-name, tot-balance)
      )
```

b. Using a nested query in a **having** clause.

```
select branch-name, sum (balance)
from account
group by branch-name
having sum (balance) ¡
      ( select avg (tot-balance)
      from ( select branch-name, sum (balance)
            from account
            group by branch-name) as branch-total(branch-name, tot-balance)
      )
```

4.11 Suppose that we have a relation *marks*(*student-id*, *score*) and we wish to assign grades to students based on the score as follows: grade F if $score < 40$, grade C if $40 \leq score < 60$, grade B if $60 \leq score < 80$, and grade A if $80 \leq score$. Write SQL queries to do the following:

a. Display the grade for each student, based on the *marks* relation.
b. Find the number of students with each grade.

Answer: We use the **case** operation provided by SQL-92:

a. To display the grade for each student:

```
select student-id,
        (case
                when score < 40 then 'F',
                when score < 60 then 'C',
                when score < 80 then 'B',
                else 'A'
        end) as grade
from marks
```

b. To find the number of students with each grade we use the following query, where *grades* is the result of the query given as the solution to part 0.a.

```
select grade, count(student-id)
from grades
group by grade
```

4.12 SQL-92 provides an n-ary operation called **coalesce**, which is defined as follows: **coalesce**$(A_1, A_2, \ldots, A_n)$ returns the first nonnull A_i in the list $A_1, A_2, \ldots, A_n$, and returns null if all of $A_1, A_2, \ldots, A_n$ are null. Show how to express the **coalesce** operation using the **case** operation.

Answer:

```
case
    when A_1 is not null then A_1
    when A_2 is not null then A_2
    ...
    when A_n is not null then A_n
    else null
end
```

4.13 Let a and b be relations with the schemas $A(name, address, title)$ and $B(name, address, salary)$, respectively. Show how to express a **natural full outer join** b using the **full outer join** operation with an **on** condition and the **coalesce** operation. Make sure that the result relation does not contain two copies of the attributes *name* and *address*, and that the solution is correct even if some tuples in a and b have null values for attributes *name* or *address*.

Answer:

```
select coalesce(a.name, b.name) as name,
        coalesce(a.address, b.address) as address,
        a.title,
        b.salary
from  a full outer join b on a.name = b.name and
                             a.address = b.address
```

4.14 Give an SQL schema definition for the employee database of Figure 4.13. Choose an appropriate domain for each attribute and an appropriate primary key for each relation schema.

Answer:

```
create domain    company-names char(20)
```

```
create domain   city-names char(30)
create domain   person-names char(20)

create table    employee
(employee-name  person-names,
 street         char(30),
 city           city-names,
 primary key    (employee-name))

create table    works
(employee-name  person-names,
 company-name   company-names,
 salary         numeric(8, 2),
 primary key    (employee-name))

create table    company
(company-name   company-names,
 city           city-names,
 primary key    (company-name))

create table    manages
(employee-name  person-names,
 manager-name   person-names,
 primary key    (employee-name))
```

4.15 Write **check** conditions for the schema you defined in Exercise 4.14 to ensure that:

a. Every employee works for a company located in the same city as the city in which the employee lives.

b. No employee earns a salary higher than that of his manager.

Answer:

a. check condition for the *works* table:-

```
check((employee-name, company-name) in
        (select  e.employee-name, c.company-name
         from    employee e, company c
         where   e.city = c.city
        )
)
```

b. check condition for the *works* table:-

```
check(
        salary < all
                (select   manager-salary
                 from     (select manager-name, manages.employee-name as emp-name,
                                  salary as manager-salary
                           from works, manages
                           where works.employee-name = manages.manager-name)
                 where    employee-name = emp-name
                )
)
```

The solution is slightly complicated because of the fact that inside the **select** expression's scope, the outer *works* relation into which the insertion is being performed is inaccessible. Hence the renaming of the *employee-name* attribute to *emp-name*. Under these circumstances, it is more natural to use assertions, which are introduced in Chapter 6.

4.16 Describe the circumstances in which you would choose to use embedded SQL rather than SQL alone or only a general-purpose programming language.

Answer: Writing queries in SQL is typically much easier than coding the same queries in a general-purpose programming language. However not all kinds of queries can be written in SQL. Also nondeclarative actions such as printing a report, interacting with a user, or sending the results of a query to a graphical user interface cannot be done from within SQL. Under circumstances in which we want the best of both worlds, we can choose embedded SQL or dynamic SQL, rather than using SQL alone or using only a general-purpose programming language.

Embedded SQL has the advantage of programs being less complicated since it avoids the clutter of the ODBC or JDBC function calls, but requires a specialized preprocessor.

CHAPTER 5

Other Relational Languages

In this chapter we study two additional relational languages, QBE and Datalog. QBE, based on the domain relational calculus, forms the basis for query languages supported by a large number of database systems designed for personal computers, such as Microsoft Access, FoxPro, etc. Unfortunately there is no standard for QBE; our coverage is based on the original description of QBE. The description here will have to be supplemented by material from the user guides of the specific database system being used. One of the points to watch out for is the precise semantics of aggregate operations, which is particularly non-standard.

The Datalog language has several similarities to Prolog, which some students may have studied in other courses. Datalog differs from Prolog in that its semantics is purely declarative, as opposed to the operational semantics of Prolog. It is important to emphasize the differences, since the declarative semantics enables the use of efficient query evaluation strategies. There are several implementations of Datalog available in the public domain, such as the Coral system from the University of Wisconsin – Madison, and XSB from the State University of New York, Stony Brook, which can be used for programming exercises. The Coral system also supports complex objects such as nested relations (covered later in Chapter 9). See the Tools section at the end of Chapter 5 for the URLs of these systems.

Changes from 3rd edition:

The syntax and semantics of QBE aggregation and update have been changed to simplify the semantics and to remove some ambiguities in the earlier semantics. The version of QBE supported by Microsoft Access has been covered briefly. Quel has been dropped.

Exercises

5.1 Consider the insurance database of Figure 5.14, where the primary keys are underlined. Construct the following QBE queries for this relational-database.

a. Find the total number of people who owned cars that were involved in accidents in 1989.
b. Find the number of accidents in which the cars belonging to "John Smith" were involved.
c. Add a new accident to the database; assume any values for required attributes.
d. Delete the Mazda belonging to "John Smith."
e. Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

Answer: The *participated* relation relates car(s) and accidents. Assume the *date* attribute is of the form "YYYY-MM-DD".

a. Find the total number of people who owned cars that were involved in accidents in 1989.

accident	*report-number*	*date*	*location*
	_*report*	_*date*	

participated	*driver-id*	*car*	*report-number*	damage-amount
	P.CNT.UNQ.ALL		_*report*	

conditions
_*date* = ($\geq$ 1989-00-00 **and** $\leq$ 1989-12-31)

b. Find the number of accidents in which the cars belonging to "John Smith" were involved.

person	*driver-id*	*name*	*address*
	_*driver*	John Smith	

participated	*driver-id*	*car*	*report-number*	*damage-amount*
	_*driver*		P.CNT.ALL	

c. Add a new accident to the database; assume any values for required attributes.

We assume that the driver was "Williams", although it could have been someone else. Also assume that "Williams" has only one Toyota.

accident	*report-number*	*date*	*location*
I.	4007	1997-01-01	Berkeley

person (*driver-id*, *name*, *address*)
car (*license*, *model*, *year*)
accident (*report-number*, *date*, *location*)
owns (*driver-id*, *license*)
participated (*driver-id*, *car*, *report-number*, *damage-amount*)

Figure 5.14. Insurance database.

participated	*driver-id*	*car*	*report-number*	*damage-amount*
I.	_*driver*	_*license*	4007	3000

owns	*driver-id*	*license*
	_*driver*	_*license*

car	*license*	*year*	*model*
	_*license*	_*year*	Toyota

person	*driver-id*	*name*	*address*
	_*driver*	Williams	

d. Delete the car "Mazda" that belongs to "John Smith."

person	*driver-id*	*name*	*address*
	_*driver*	John Smith	

owns	*driver-id*	*license*
	_*driver*	_*license*

car	*license*	*year*	*model*
D.	_*license*		Mazda

e. Update the damage amount for the car with license number "AABB2000" in the accident with report number "AR2197" to $3000.

owns	*driver-id*	*license*
	_*driver*	"AABB2000"

participated	*driver-id*	*car*	*report-number*	*damage-amount*
	_*driver*		"AR2197"	U.3000

5.2 Consider the employee database of Figure 5.15. Give expressions in QBE, and Datalog for each of the following queries:

a. Find the names of all employees who work for First Bank Corporation.

b. Find the names and cities of residence of all employees who work for First Bank Corporation.

c. Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000 per annum.

d. Find all employees who live in the same city as the company for which they work is located.

e. Find all employees who live in the same city and on the same street as their managers.

f. Find all employees in the database who do not work for First Bank Corporation.

g. Find all employees who earn more than every employee of Small Bank Corporation.

h. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

Answer:

a. Find the names of all employees who work for First Bank Corporation.

i.

works	*person-name*	*company-name*	*salary*
	P._x	First Bank Corporation	

ii. *query(X)* :- *works(X, "First Bank Corporation", Y)*

b. Find the names and cities of residence of all employees who work for First Bank Corporation.

i.

works	*person-name*	*company-name*	*salary*
	_x	First Bank Corporation	

employee	*person-name*	*street*	*city*
	P._x		P._y

ii. $query\,(X,\,Y)$:- $employee\,(X,\,Z,\,Y),\ works\,(X,$ "First Bank Corporation", $W)$

c. Find the names, street addresses, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000 per annum.

If people may work for several companies, the following solutions will only list those who earn more than $10,000 per annum from "First Bank Corporation" alone.

i.

employee	*person-name*	*street*	*city*
	P._x	P._y	P._z

works	*person-name*	*company-name*	*salary*
	_*x*	First Bank Co	> 10000

ii.

$$query\,(X,\,Y,\,Z)\text{ :- }lives\,(X,\,Y,\,Z),\;works\,(X,\,\text{“First Bank Corporation”},\,W),\;W\;>\;10000$$

d. Find all employees who live in the city where the company for which they work is located.

i.

employee	*person-name*	*street*	*city*
	P._*x*		_*y*

works	*person-name*	*company-name*	*salary*
	_*x*	_*c*	

company	*company-name*	*city*
	_*c*	_*y*

ii. $query\,(X)$:- $employee\,(X,\,Y,\,Z),\;works\,(X,\,V,\,W),\;company\,(V,\,Z)$

e. Find all employees who live in the same city and on the same street as their managers.

i.

employee	*person-name*	*street*	*city*
	P._*x*	_*s*	_*c*
	_*y*	_*s*	_*c*

manages	*person-name*	*manager − name*
	_*x*	_*y*

ii. $query\,(X)$:- $lives\,(X,\,Y,\,Z),\;manages\,(X,\,V),\;lives\,(V,\,Y,\,Z)$

f. Find all employees in the database who do not work for First Bank Corporation.

The following solutions assume that all people work for exactly one company.

i.

works	*person-name*	*company-name*	*salary*
	P._*x*	¬ First Bank Co	

ii. $query\,(X)$:- $works\,(X,\,Y,\,Z),\;Y \neq$ “First Bank Corporation”

If one allows people to appear in the database (e.g. in *employee*) but not appear in *works*, or if people may have jobs with more than one company, the solutions are slightly more complicated. They are given below :-

i.

employee	*person-name*	*street*	*city*
	P._x		

works	*person-name*	*company-name*	*salary*
¬	_x	First Bank Corporation	

ii.

$query\,(X)$:- $employee\,(X,\ Y,\ Z),\ \neg\, p1\,(X)$
$p1\,(X)$:- $works(X,$ "First Bank Corporation", $W)$

g. Find all employees who earn more than every employee of Small Bank Corporation.

The following solutions assume that all people work for at most one company.

i.

works	*person-name*	*company-name*	*salary*
	P._x	Small Bank Co	_y > MAX.ALL._y

or

works	*person-name*	*company-name*	*salary*
¬	P._x	Small Bank Co	_y > _y

ii.

$query\,(X)$:- $works\,(X,\ Y,\ Z),\ \neg\, p\,(X)$
$p\,(X)$:- $works(X, C, Y1),\ works\,(V,$ "Small Bank Corporation", $Y),\ Y > Y1$

h. Assume that the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

Note: Small Bank Corporation will be included in each answer.

i.

located-in	*company-name*	*city*
	Small Bank Corporation	_x
	P._c	_y
	Small Bank Corporation	_y

conditions
CNT.ALL._y = CNT.ALL._x

ii.

query(X) :- company(X, C), not p(X)
p(X) :- company(X, C1), company("Small Bank Corporation", C2), not company(X, C2)

5.3 Consider the relational database of Figure 5.15. where the primary keys are underlined. Give expressions in QBE for each of the following queries:

a. Find all employees who earn more than the average salary of all employees of their company.
b. Find the company that has the most employees.
c. Find the company that has the smallest payroll.
d. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

Answer:

a. Find all employees who earn more than the average salary of all employees of their company.

The following solution assumes that all people work for at most one company.

works	*person-name*	*company-name*	*salary*
	P.	_*y*	_*x*
		_*y*	_*z*

conditions
_*x* > AVG.ALL._*z*

b. Find the company that has the most employees.

works	*person-name*	*company-name*	*salary*
	_*x*	P.G.	
	_*y*	G.	

conditions
CNT.UNQ._*x* ≥ MAX.CNT.UNQ.ALL._*y*

c. Find the company that has the smallest payroll.

employee (*person-name*, *street*, *city*)
works (*person-name*, *company-name*, *salary*)
company (*company-name*, *city*)
manages (*person-name*, *manager-name*)

Figure 5.15. Employee database.

works	*person-name*	*company-name*	*salary*
		P.G.	_*x*
		G.	_*y*

conditions
SUM.ALL._*x* ≤ MIN.SUM.ALL._*y*

d. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

works	*person-name*	*company-name*	*salary*
		P.G.	_*x*
		First Bank Corporation	_*y*

conditions
AVG.ALL._*x* > AVG.ALL._*y*

5.4 Consider the relational database of Figure 5.15. Give expressions in QBE for each of the following queries:

a. Modify the database so that Jones now lives in Newtown.
b. Give all employees of First Bank Corporation a 10 percent raise.
c. Give all managers in the database a 10 percent raise.
d. Give all managers in the database a 10 percent raise, unless the salary would be greater than $100,000. In such cases, give only a 3 percent raise.
e. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

Answer: The solutions assume that each person has only one tuple in the *employee* relation. The solutions to parts 0.c and 0.d assume that each person works for at most one company.

a. Modify the database so that Jones now lives in Newtown.

employee	*person-name*	*street*	*city*
	Jones		U.Newtown

b. Give all employees of First Bank Corporation a 10-percent raise.

works	*person-name*	*company-name*	*salary*
		First Bank Corporation	_*x*
U.			_*x* * 1.1

c. Give all managers in the database a 10-percent raise.

manages	*person-name*	*manager-name*
		_*x*

works	*person-name*	*company-name*	*salary*
	_x		_y
U.			_y * 1.1

d. Give all managers in the database a 10-percent raise, unless the salary would be greater than $100,000. In such cases, give only a 3-percent raise. Two separate update operations must be performed. Each update operation has its own set of skeleton tables.
First update:

manages	*person-name*	*manager-name*
		_x

works	*person-name*	*company-name*	*salary*
	_x		_y
U.			_y * 1.03

conditions
_y > 100000/1.1

Second update:

manages	*person-name*	*manager-name*
		_x

works	*person-name*	*company-name*	*salary*
	_x		_y
U.			_y * 1.1

conditions
_y ≤ 100000/1.1

e. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

works	*person-name*	*company-name*	*salary*
D.		Small Bank Co	

5.5 Let the following relation schemas be given:

$$R = (A, B, C)$$
$$S = (D, E, F)$$

Let relations *r*(*R*) and *s*(*S*) be given. Give expressions in QBE, and Datalog equivalent to each of the following queries:

a. $\Pi_A(r)$
b. $\sigma_{B=17}(r)$
c. $r \times s$

d. $\Pi_{A,F}\,(\sigma_{C=D}(r \times s))$

Answer:

a. $\Pi_A(r)$

i.

r	A	B	C
	P.		

ii. $query\,(X)$:- $r\,(X, Y, Z)$

b. $\sigma_{B=17}\,(r)$

i.

r	A	B	C
P.		17	

ii. $query\,(X, Y, Z)$:- $r\,(X, Y, Z),\ Y = 17$

c. $r \times s$

i.

$result$	A	B	C	D	E	F
P.	$\_a$	$\_b$	$\_c$	$\_d$	$\_e$	$\_f$

r	A	B	C
	$\_a$	$\_b$	$\_c$

s	D	E	F
	$\_d$	$\_e$	$\_f$

ii. $query\,(X, Y, Z, U, V, W)$:- $r\,(X, Y, Z),\ s\,(U, V, W)$

d. $\Pi_{A,F}\,(\sigma_{C=D}(r \times s))$

i.

$result$	A	F
P.	$\_a$	$\_f$

r	A	B	C
	$\_a$		$\_c$

s	D	E	F
	$\_c$		$\_f$

ii. $query\,(X, Y)$:- $r\,(X, V, W),\ s\,(W, Z, Y)$

5.6 Let $R = (A, B, C)$, and let r_1 and r_2 both be relations on schema R. Give expressions in QBE, and Datalog equivalent to each of the following queries:

a. $r_1 \cup r_2$
b. $r_1 \cap r_2$
c. $r_1 - r_2$
d. $\Pi_{AB}(r_1) \bowtie \Pi_{BC}(r_2)$

Answer:

a. $r_1 \cup r_2$
i.

result	A	B	C
P.	_a	_b	_c
P.	_d	_e	_f

$r1$	A	B	C
	_a	_b	_c

$r2$	A	B	C
	_d	_e	_f

ii.
query(X, Y, Z) :- r_1 (X, Y, Z)
query(X, Y, Z) :- r_2 (X, Y, Z)

b. $r_1 \cap r_2$
i.

$r1$	A	B	C
P.	_a	_b	_c

$r2$	A	B	C
	_a	_b	_c

ii. *query*(X, Y, Z) :- r_1 (X, Y, Z), r_2 (X, Y, Z)

c. $r_1 - r_2$
i.

$r1$	A	B	C
P.	_a	_b	_c

$r2$	A	B	C
$\neg$	$\_a$	$\_b$	$\_c$

ii. *query(X, Y, Z)* :- r_1 *(X, Y, Z), not* r_2 *(X, Y, Z)*

d. $\Pi_{AB}(r_1) \bowtie \Pi_{BC}(r_2)$

i.

$result$	A	B	C
P.	$\_a$	$\_b$	$\_c$

$r1$	A	B	C
	$\_a$	$\_b$	

$r2$	A	B	C
		$\_b$	$\_c$

ii. *query(X, Y, Z)* :- r_1 *(X, Y, V),* r_2 *(W, Y, Z)*

5.7 Let $R = (A, B)$ and $S = (A, C)$, and let $r(R)$ and $s(S)$ be relations. Write expressions in QBE and Datalog for each of the following queries:

a. $\{< a > \mid \exists\, b\, (< a, b > \in r \wedge b = 17)\}$
b. $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s\}$
c. $\{< a > \mid \exists\, c\, (< a, c > \in s \wedge \exists\, b_1, b_2\, (< a, b_1 > \in r \wedge < c, b_2 > \in r \wedge b_1 > b_2))\}$

Answer:

a. $\{< a > \mid \exists\, b\, (< a, b > \in r \wedge b = 17)\}$

i.

r	A	B
	P.	17

ii. *query* (X) :- $r\,(X,\, 17)$

b. $\{< a, b, c > \mid < a, b > \in r \wedge < a, c > \in s\}$

i.

r	A	B
	$\_a$	$\_b$

s	A	C
	$\_a$	$\_c$

$result$	A	B	C
P.	$\_a$	$\_b$	$\_c$

ii. *query(X, Y, Z)* :- *r(X, Y), s(X, Z)*

c. $\{< a > \mid \exists\, c\, (< a, c > \in\ s \wedge \exists\, b_1, b_2\, (< a, b_1 > \in\ r \wedge\ < c, b_2 > \in\ r \wedge\ b_1 > b_2))\}$

i.

r	A	B
	$\_a$	$> \_s$
	$\_c$	$\_s$

s	A	C
	P.$\_a$	$\_c$

ii. $query\,(X)$:- $s\,(X,\, Y),\ r\,(X,\, Z),\ r\,(Y,\, W),\ Z\ >\ W$

5.8 Consider the relational database of Figure 5.15. Write a Datalog program for each of the following queries:

a. Find all employees who work (directly or indirectly) under the manager "Jones".

b. Find all cities of residence of all employees who work (directly or indirectly) under the manager "Jones".

c. Find all pairs of employees who have a (direct or indirect) manager in common.

d. Find all pairs of employees who have a (direct or indirect) manager in common, and are at the same number of levels of supervision below the common manager.

Answer:

a. Find all employees who work (directly or indirectly) under the manager "Jones".

$query\,(X)$:- $p\,(X)$
$p\,(X)$:- $manages\,(X,$ "Jones")
$p\,(X)$:- $manages\,(X,\, Y),\ p\,(Y)$

b. Find all cities of residence of all employees who work (directly or indirectly) under the manager "Jones".

query(X, C) :- *p(X), employee(X, S, C)*
p(X) :- *manages(X, "Jones")*
p(X) :- *manages(X, Y), p(Y)*

c. Find all pairs of employees who have a (direct or indirect) manager in common.

query(X, Y) :- *p(X, W)*, *p(Y, W)*
p(X, Y) :- *manages(X, Y)*
p(X, Y) :- *manages(X, Z)*, *p(Z, Y)*

d. Find all pairs of employees who have a (direct or indirect) manager in common, and are at the same number of levels of supervision below the common manager.

query(X, Y) :- *p(X, Y)*
p(X, Y) :- *manages(X, Z)*, *manages(Y, Z)*
p(X, Y) :- *manages(X, V)*, *manages(Y, W)*, *p(V, W)*

5.9 Write an extended relational-algebra view equivalent to the Datalog rule

$$p(A, C, D) \text{ :- } q1(A, B),\ q2(B, C),\ q3(4, B),\ D = B + 1\ .$$

Answer: Let us assume that $q1$, $q2$ and $q3$ are instances of the schema $(A1, A2)$. The relational algebra view is

create view P **as**
$\Pi_{q1.A1,q2.A2,q1.A2+1}(\sigma_{q3.A1=4\ \wedge\ q1.A2=q2.A1\ \wedge\ q1.A2=q3.A2}(q1\ \times\ q2\ \times\ q3))$

5.10 Describe how an arbitrary Datalog rule can be expressed as an extended relational-algebra view.

Answer: A Datalog rule has two parts, the *head* and the *body*. The body is a comma separated list of *literals*. A *positive literal* has the form $p(t_1,\ t_2, \ldots,\ t_n\ )$ where p is the name of a relation with n attributes, and $t_1,\ t_2, \ldots,\ t_n$ are either constants or variables. A *negative literal* has the form $\neg p(t_1,\ t_2, \ldots,\ t_n)$ where p has n attributes. In the case of arithmetic literals, p will be an arithmetic operator like $>$, $=$ etc.

We consider only safe rules; see Section 5.2.4 for the definition of safety of Datalog rules. Further, we assume that every variable that occurs in an arithmetic literal also occurs in a positive non-arithmetic literal.

Consider first a rule without any negative literals. To express the rule as an extended relational-algebra view, we write it as a join of all the relations referred to in the (positive) non-arithmetic literals in the body, followed by a selection. The selection condition is a conjunction obtained as follows. If $p_1\ (X,\ Y),\ p_2\ (Y,\ Z)$ occur in the body, where p_1 is of the schema $(A,\ B)$ and p_2 is of the schema $(C,\ D)$, then $p_1.B\ =\ p_2.C$ should belong to the conjunction. The arithmetic literals can then be added to the condition.

As an example, the Datalog query

query(X, Y) :- *works(X, C, S1)*, *works(Y, C, S2)*, *S1 > S2*, *manages(X, Y)*

becomes the following relational-algebra expression:

$$E_1 = \sigma_{(w1.company\text{-}name = w2.company\text{-}name \wedge w1.salary > w2.salary \wedge manages.person\text{-}name = w1.person\text{-}name \wedge manages.manager\text{-}name = w2.person\text{-}name)} (\rho_{w1}(works) \times \rho_{w2}(works) \times manages)$$

Now suppose the given rule has negative literals. First suppose that there are no constants in the negative literals; recall that all variables in a negative literal must also occur in a positive literal. Let $\neg q(X, Y)$ be the first negative literal, and let it be of the schema (E, F). Let E_i be the relational algebra expression obtained after all positive and arithmetic literals have been handled. To handle this negative literal, we generate the expression

$$E_j = E_i \bowtie (\Pi_{A_1, A_2}(E_i) - q)$$

where A_1 and A_2 are the attribute names of two columns in E_i which correspond to X and Y respectively.

Now let us consider constants occurring in a negative literal. Consider a negative literal of the form $\neg q(a, b, Y)$ where a and b are constants. Then, in the above expression defining E_j we replace q by $\sigma_{A_1 = a \wedge A_2 = b}(q)$.

Proceeding in a similar fashion, the remaining negative literals are processed, finally resulting in an expression E_w.

Finally the desired attributes are projected out of the expression. The attributes in E_w corresponding to the variables in the head of the rule become the projection attributes.

Thus our example rule finally becomes the view:-

create view *query* **as**

$$\Pi_{w1.person\text{-}name,\, w2.person-name}(E_2)$$

If there are multiple rules for the same predicate, the relational-algebra expression defining the view is the union of the expressions corresponding to the individual rules.

The above conversion can be extended to handle rules that satisfy some weaker forms of the safety conditions, and where some restricted cases where the variables in arithmetic predicates do not appear in a positive non-arithmetic literal.

CHAPTER 6

Integrity and Security

This chapter presents several types of integrity constraints, including domain constraints, referential integrity constraints, assertions and triggers, as well as security and authorization. Referential integrity constraints, and domain constraints are an important aspect of the specification of a relational database design. Assertions are seeing increasing use. Triggers are widely used, although each database supports its own syntax and semantics for triggers; triggers were standardized as part of SQL:1999, and we can expect databases to provide support for SQL:1999 triggers.

Functional dependencies are now taught as part of normalization instead of being part of the integrity constraints chapter as they were in the 3^{rd} edition. The reason for the change is that they are used almost exclusively in database design, and no database system to our knowledge supports functional dependencies as integrity constraints. Covering them in the context of normalization helps motivate students to spend the effort to understand the intricacies of reasoning with functional dependencies.

Security is a major topic in its own right. Since any system is only as secure as its weakest component, a system builder must consider all aspects of security. This chapter focuses only on those security issues that are specific to databases. In an advanced course, this material can be supplemented by discussion of security issues in operating systems and in distributed systems.

Changes from 3^{rd} edition:

Trigger coverage is now based on the SQL:1999 standard. At the time of publication of the 3^{rd} edition, triggers had not been standardized. The notion of roles for authorization has been introduced in this edition, now that it is a part of the SQL:1999 standard. Coverage of encryption has been updated to cover recent developments.

Exercises

6.1 Complete the SQL DDL definition of the bank database of Figure 6.2 to include the relations *loan* and *borrower*.
Answer:

```
create table loan
    (loan-number     char(10),
     branch-name     char(15),
     amount          integer,
     primary key (loan-number),
     foreign key (branch-name) references branch)


create table borrower
    (customer-name  char(20),
     loan-number    char(10),
     primary key (customer-name, loan-number),
     foreign key (customer-name) references customer,
     foreign key (loan-number) references loan)
```

Declaring the pair *customer-name, loan-number* of relation *borrower* as primary key ensures that the relation does not contain duplicates.

6.2 Consider the following relational database:

employee (*employee-name*, *street*, *city*)
works (*employee-name*, *company-name*, *salary*)
company (*company-name*, *city*)
manages (*employee-name*, *manager-name*)

Give an SQL DDL definition of this database. Identify referential-integrity constraints that should hold, and include them in the DDL definition.
Answer:

```
create table employee
    (person-name    char(20),
     street         char(30),
     city           char(30),
     primary key (person-name) )
```

create table *works*
(*person-name* char(20),
company-name char(15),
salary integer,
primary key (*person-name*),
foreign key (*person-name*) **references** *employee*,
foreign key (*company-name*) **references** *company*)

create table *company*
(*company-name* char(15),
city char(30),
primary key (*company-name*))

create table *manages*
(*person-name* char(20),
manager-name char(20),
primary key (*person-name*),
foreign key (*person-name*) **references** *employee*,
foreign key (*manager-name*) **references** *employee*)

Note that alternative datatypes are possible. Other choices for **not null** attributes may be acceptable.

6.3 Referential-integrity constraints as defined in this chapter involve exactly two relations. Consider a database that includes the following relations:

salaried-worker (*name, office, phone, salary*)
hourly-worker (*name, hourly-wage*)
address (*name, street, city*)

Suppose that we wish to require that every name that appears in *address* appear in either *salaried-worker* or *hourly-worker*, but not necessarily in both.

a. Propose a syntax for expressing such constraints.
b. Discuss the actions that the system must take to enforce a constraint of this form.

Answer:

a. For simplicity, we present a variant of the SQL syntax. As part of the **create table** expression for *address* we include

foreign key (*name*) **references** *salaried-worker* or *hourly-worker*

b. To enforce this constraint, whenever a tuple is inserted into the *address* relation, a lookup on the *name* value must be made on the *salaried-worker* relation and (if that lookup failed) on the *hourly-worker* relation (or vice-versa).

6.4 SQL allows a foreign-key dependency to refer to the same relation, as in the following example:

```
create table manager
    (employee-name    char(20),
     manager-name     char(20),
     primary key employee-name,
     foreign key (manager-name) references manager
                                    on delete cascade )
```

Here, *employee-name* is a key to the table *manager*, meaning that each employee has at most one manager. The foreign-key clause requires that every manager also be an employee. Explain exactly what happens when a tuple in the relation *manager* is deleted.

Answer: The tuples of all employees of the manager, at all levels, get deleted as well! This happens in a series of steps. The initial deletion will trigger deletion of all the tuples corresponding to direct employees of the manager. These deletions will in turn cause deletions of second level employee tuples, and so on, till all direct and indirect employee tuples are deleted.

6.5 Suppose there are two relations r and s, such that the foreign key B of r references the primary key A of s. Describe how the trigger mechanism can be used to implement the **on delete cascade** option, when a tuple is deleted from s.

Answer: We define triggers for each relation whose primary-key is referred to by the foreign-key of some other relation. The trigger would be activated whenever a tuple is deleted from the referred-to relation. The action performed by the trigger would be to visit all the referring relations, and delete all the tuples in them whose foreign-key attribute value is the same as the primary-key attribute value of the deleted tuple in the referred-to relation. These set of triggers will take care of the **on delete cascade** operation.

6.6 Write an assertion for the bank database to ensure that the assets value for the Perryridge branch is equal to the sum of all the amounts lent by the Perryridge branch.

Answer: The assertion-name is arbitrary. We have chosen the name *perry*. Note that since the assertion applies only to the Perryridge branch we must restrict attention to only the Perryridge tuple of the *branch* relation rather than writing a constraint on the entire relation.

```
create assertion perry check
    (not exists (select *
                 from branch
                 where branch-name = 'Perryridge' and
                       assets ≠ (select sum (amount)
                                 from loan
                                 where branch-name = 'Perryridge')))
```

6.7 Write an SQL trigger to carry out the following action: On **delete** of an account, for each owner of the account, check if the owner has any remaining accounts, and if she does not, delete her from the *depositor* relation.
Answer:

```
create trigger check-delete-trigger after delete on account
referencing old row as orow
for each row
delete from depositor
where depositor.customer-name not in
    ( select customer-name from depositor
    where account-number <> orow.account-number )
end
```

6.8 Consider a view *branch-cust* defined as follows:

```
create view branch-cust as
    select branch-name, customer-name
    from depositor, account
    where depositor.account-number = account.account-number
```

Suppose that the view is *materialized*, that is, the view is computed and stored. Write active rules to *maintain* the view, that is, to keep it up to date on insertions to and deletions from *depositor* or *account*. Do not bother about updates.
Answer: For inserting into the materialized view *branch-cust* we must set a database trigger on an insert into *depositor* and *account*. We assume that the database system uses *immediate* binding for rule execution. Further, assume that the current version of a relation is denoted by the relation name itself, while the set of newly inserted tuples is denoted by qualifying the relation name with the prefix – **inserted**.

The active rules for this insertion are given below –

```
define trigger insert_into_branch-cust_via_depositor
after insert on depositor
referencing new table as inserted for each statement
insert into branch-cust
    select branch-name, customer-name
    from inserted, account
    where inserted.account-number = account.account-number

define trigger insert_into_branch-cust_via_account
after insert on account
referencing new table as inserted for each statement
insert into branch-cust
    select branch-name, customer-name
    from depositor, inserted
    where depositor.account-number = inserted.account-number
```

Note that if the execution binding was *deferred* (instead of immediate), then the result of the join of the set of new tuples of *account* with the set of new tuples of *depositor* would have been inserted by *both* active rules, leading to duplication of the corresponding tuples in *branch-cust*.

The deletion of a tuple from *branch-cust* is similar to insertion, except that a deletion from either *depositor* or *account* will cause the natural join of these relations to have a lesser number of tuples. We denote the newly deleted set of tuples by qualifying the relation name with the keyword **deleted**.

define trigger *delete_from_branch-cust_via_depositor*
after delete on *depositor*
referencing old table as *deleted* **for each statement**
delete from *branch-cust*
  select *branch-name, customer-name*
  from *deleted, account*
  where *deleted.account-number = account.account-number*

define trigger *delete_from_branch-cust_via_account*
after delete on *account*
referencing old table as *deleted* **for each statement**
delete from *branch-cust*
  select *branch-name, customer-name*
  from *depositor, deleted*
  where *depositor.account-number = deleted.account-number*

6.9 Make a list of security concerns for a bank. For each item on your list, state whether this concern relates to physical security, human security, operating-system security, or database security.

Answer: Let us consider the problem of protecting our sample bank database. Some security measures at each of the four levels are mentioned below -

a. Physical level - The system from which the relations can be accessed and modified should be placed in a locked, well-guarded, and impregnable room.

b. Human level - A proper key transfer policy should be enforced for restricting access to the "system room" mentioned above. Passwords for gaining access to the database should be known only to trusted users.

c. Operating System level - Login passwords should be difficult to guess and they should be changed regularly. No user should be able to gain unauthorized access to the system due to a software bug in the operating system.

d. Database System level - The users should be authorized access only to relevant parts of the database. For example, a bank teller should be allowed to modify values for the customer's balance, but not for her own salary.

6.10 Using the relations of our sample bank database, write an SQL expression to define the following views:

a. A view containing the account numbers and customer names (but not the balances) for all accounts at the Deer Park branch.

b. A view containing the names and addresses of all customers who have an account with the bank, but do not have a loan.

c. A view containing the name and average account balance of every customer of the Rock Ridge branch.

Answer:

a.

```
create view deer-park as
select account-number, customer-name
from depositor, account
where branch-name = 'Deer Park' and
      depositor.account-number = account.account-number
```

b.

```
create view no-debt as
select * from customer
where customer-name in
        (select customer-name
         from depositor)
        minus
        (select customer-name
         from borrower)
```

c.

```
create view avg-bal as
select customer-name, avg(balance)
from depositor, account
where depositor.account-number = account.account-number
        and branch-name = 'Rock Ridge'
group by customer-name
```

6.11 For each of the views that you defined in Exercise 6.10, explain how updates would be performed (if they should be allowed at all). *Hint*: See the discussion of views in Chapter 3.

Answer: To insert (*account-number*, *name*) into the view *deer-park* we insert the tuple (Deer Park, *account-number*, *null*) into the *account* relation and the tuple (*name*, *account-number*) into the *depositor* relation.

Updates to the views *no-debt* and *avg-bal* present serious problems. If we insert into the *no-debt* view, the system must reject the insertion if the customer has a loan. The overhead of updating through this view is so high that most systems would disallow update. The *avg-bal* view cannot be updated since the result of an aggregate operation depends on several tuples, not just one.

6.12 In Chapter 3, we described the use of views to simplify access to the database by users who need to see only part of the database. In this chapter, we described the use of views as a security mechanism. Do these two purposes for views ever conflict? Explain your answer.

Answer: Usually, a well-designed view and security mechanism can avoid con-

flicts between ease of access and security. However, as the following example shows, the two purposes do conflict in case the mechanisms are not designed carefully.

Suppose we have a database of employee data and a user whose view involves employee data for employees earning less than $10,000. If this user inserts employee Jones, whose salary is $9,000, but accidentally enters $90,000, several existing database systems will accept this update as a valid update through a view. However, the user will be denied access to delete this erroneous tuple by the security mechanism.

6.13 What is the purpose of having separate categories for index authorization and resource authorization?

Answer: Index and resource authorization should be special categories to allow certain users to create relations (and the indices to operate on them) while preventing these time-consuming and schema-changing operations from being available to many users. Separating index and resource authorization allows a user to build an index on existing relations, say, for optimization purposes, but allows us to deny that user the right to create new relations.

6.14 Database systems that store each relation in a separate operating-system file may use the operating system's security and authorization scheme, instead of defining a special scheme themselves. Discuss an advantage and a disadvantage of such an approach.

Answer: Database systems have special requirements which are typically more refined than most operating systems. For example, a single user may have different privileges on different files throughout the system, including changing indices and attributes which file systems typically don't monitor. The advantage of using the operating system's security mechanism is that it simplifies the database system and can be used for simple (read/write) security measures.

6.15 What are two advantages of encrypting data stored in the database?

Answer:

a. Encrypted data allows authorized users to access data without worrying about other users or the system administrator gaining any information.

b. Encryption of data may simplify or even strengthen other authorization mechanisms. For example, distribution of the cryptographic key amongst only trusted users is both, a simple way to control read access, and an added layer of security above that offered by views.

6.16 Perhaps the most important data items in any database system are the passwords that control access to the database. Suggest a scheme for the secure storage of passwords. Be sure that your scheme allows the system to test passwords supplied by users who are attempting to log into the system.

Answer: A scheme for storing passwords would be to encrypt each password, and then use a hash index on the user-id. The user-id can be used to easily access the encrypted password. The password being used in a login attempt is then encrypted and compared with the stored encryption of the correct password. An

advantage of this scheme is that passwords are not stored in clear text and the code for decryption need not even exist!

CHAPTER 7

Relational-Database Design

This chapter presents the principles of relational database design. Undergraduates frequently find this chapter difficult. It is acceptable to cover only Sections 7.1, 7.2 and 7.4 for classes that find the material particularly difficult. However, a careful study of data dependencies and normalization is a good way to introduce students to the formal aspects of relational database theory.

There are many ways of stating the definitions of the normal forms. We have chosen a style which we think is the easiest to present and which most clearly conveys the intuition of the normal forms.

Changes from 3rd edition:
There are many changes to this chapter from the 3rd edition. 1NF is now defined formally. Functional dependencies are now covered in this chapter, instead of Chapter 6. The reason is that normalization provides the real motivation for functional dependencies, since they are used primarily for normalization.

We have described a simplified procedure for functional dependency inference based on attribute closure, and provided simplified procedures to test for normal forms.

Coverage of multivalued dependency theory and normal forms beyond 4NF (that is, PJNF and DKNF) has been moved into Appendix C (which is available on the web, not in the print form of the book).

The process of practical relational schema design has been described in significantly more detail, along with some design problems that are not caught by the usual normalization process.

Exercises

7.1 Explain what is meant by *repetition of information* and *inability to represent information*. Explain why each of these properties may indicate a bad relational-database design.

Answer:

- Repetition of information is a condition in a relational database where the values of one attribute are determined by the values of another attribute in the same relation, and both values are repeated throughout the relation. This is a bad relational database design because it increases the storage required for the relation and it makes updating the relation more difficult.
- Inability to represent information is a condition where a relationship exists among only a proper subset of the attributes in a relation. This is bad relational database design because all the unrelated attributes must be filled with null values otherwise a tuple without the unrelated information cannot be inserted into the relation.
- Loss of information is a condition of a relational database which results from the decomposition of one relation into two relations and which cannot be combined to recreate the original relation. It is a bad relational database design because certain queries cannot be answered using the reconstructed relation that could have been answered using the original relation.

7.2 Suppose that we decompose the schema $R = (A, B, C, D, E)$ into

$$(A, B, C)$$
$$(A, D, E).$$

Show that this decomposition is a lossless-join decomposition if the following set *F* of functional dependencies holds:

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

Answer: A decomposition $\{R_1, R_2\}$ is a lossless-join decomposition if $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$. Let $R_1 = (A, B, C)$, $R_2 = (A, D, E)$, and $R_1 \cap R_2 = A$. Since A is a candidate key (see Exercise 7.11), Therefore $R_1 \cap R_2 \rightarrow R_1$.

7.3 Why are certain functional dependencies called *trivial* functional dependencies?

Answer: Certain functional dependencies are called trivial functional dependencies because they are satisfied by all relations.

7.4 List all functional dependencies satisfied by the relation of Figure 7.21.

Answer: The nontrivial functional dependencies are: $A \rightarrow B$ and $C \rightarrow B$,

and a dependency they logically imply: $AC \rightarrow B$. There are 19 trivial functional dependencies of the form $\alpha \rightarrow \beta$, where $\beta \subseteq \alpha$. C does not functionally determine A because the first and third tuples have the same C but different A values. The same tuples also show B does not functionally determine A. Likewise, A does not functionally determine C because the first two tuples have the same A value and different C values. The same tuples also show B does not functionally determine C.

7.5 Use the definition of functional dependency to argue that each of Armstrong's axioms (reflexivity, augmentation, and transitivity) is sound.

Answer: The definition of functional dependency is: $\alpha \rightarrow \beta$ holds on R if in any legal relation $r(R)$, for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, it is also the case that $t_1[\beta] = t_2[\beta]$.

Reflexivity rule: if α is a set of attributes, and $\beta \subseteq \alpha$, then $\alpha \rightarrow \beta$. Assume $\exists\, t_1$ and t_2 such that $t_1[\alpha] = t_2[\alpha]$

$t_1[\beta] = t_2[\beta]$	since $\beta \subseteq \alpha$
$\alpha \rightarrow \beta$	definition of FD

Augmentation rule: if $\alpha \rightarrow \beta$, and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$. Assume $\exists\, t_1,\ t_2$ such that $t_1[\gamma\alpha] = t_2[\gamma\alpha]$

$t_1[\gamma] = t_2[\gamma]$	$\gamma \subseteq \gamma\alpha$
$t_1[\alpha] = t_2[\alpha]$	$\alpha \subseteq \gamma\alpha$
$t_1[\beta] = t_2[\beta]$	definition of $\alpha \rightarrow \beta$
$t_1[\gamma\beta] = t_2[\gamma\beta]$	$\gamma\beta = \gamma \cup \beta$
$\gamma\alpha \rightarrow \gamma\beta$	definition of FD

Transitivity rule: if $\alpha \rightarrow \beta$ and $\beta \rightarrow \gamma$, then $\alpha \rightarrow \gamma$.
Assume $\exists\, t_1,\ t_2$ such that $t_1[\alpha] = t_2[\alpha]$

$t_1[\beta] = t_2[\beta]$	definition of $\alpha \rightarrow \beta$
$t_1[\gamma] = t_2[\gamma]$	definition of $\beta \rightarrow \gamma$
$\alpha \rightarrow \gamma$	definition of FD

7.6 Explain how functional dependencies can be used to indicate the following:

- A one-to-one relationship set exists between entity sets *account* and *customer*.
- A many-to-one relationship set exists between entity sets *account* and *customer*.

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2
a_2	b_1	c_1
a_2	b_1	c_3

Figure 7.21. Relation of Exercise 7.4.

Answer: Let $Pk(r)$ denote the primary key attribute of relation r.

- The functional dependencies Pk(*account*) $\rightarrow Pk$ (*customer*) and Pk(*customer*) $\rightarrow Pk$(*account*) indicate a one-to-one relationship because any two tuples with the same value for account must have the same value for customer, and any two tuples agreeing on customer must have the same value for account.
- The functional dependency Pk(*account*) $\rightarrow Pk$(*customer*) indicates a many-to-one relationship since any account value which is repeated will have the same customer value, but many account values may have the same customer value.

7.7 Consider the following proposed rule for functional dependencies: If $\alpha \rightarrow \beta$ and $\gamma \rightarrow \beta$, then $\alpha \rightarrow \gamma$. Prove that this rule is *not* sound by showing a relation *r* that satisfies $\alpha \rightarrow \beta$ and $\gamma \rightarrow \beta$, but does not satisfy $\alpha \rightarrow \gamma$.
Answer: Consider the following rule: if $A \rightarrow B$ and $C \rightarrow B$, then $A \rightarrow C$. That is, $\alpha = A$, $\beta = B$, $\gamma = C$. The following relation r is a counterexample to the rule.

r:

A	B	C
a_1	b_1	c_1
a_1	b_1	c_2

Note: $A \rightarrow B$ and $C \rightarrow B$, (since no 2 tuples have the same C value, $C \rightarrow B$ is true trivially). However, it is not the case that $A \rightarrow C$ since the same A value is in two tuples, but the C value in those tuples disagree.

7.8 Use Armstrong's axioms to prove the soundness of the union rule. (*Hint*: Use the augmentation rule to show that, if $\alpha \rightarrow \beta$, then $\alpha \rightarrow \alpha\beta$. Apply the augmentation rule again, using $\alpha \rightarrow \gamma$, and then apply the transitivity rule.)
Answer: To prove that:

$$\text{if } \alpha \rightarrow \beta \text{ and } \alpha \rightarrow \gamma \text{ then } \alpha \rightarrow \beta\gamma$$

Following the hint, we derive:

$\alpha \rightarrow \beta$	given
$\alpha\alpha \rightarrow \alpha\beta$	augmentation rule
$\alpha \rightarrow \alpha\beta$	union of identical sets
$\alpha \rightarrow \gamma$	given
$\alpha\beta \rightarrow \gamma\beta$	augmentation rule
$\alpha \rightarrow \beta\gamma$	transitivity rule and set union commutativity

7.9 Use Armstrong's axioms to prove the soundness of the decomposition rule.
Answer: The decomposition rule, and its derivation from Armstrong's axioms are given below:

if $\alpha \rightarrow \beta\gamma$, then $\alpha \rightarrow \beta$ and $\alpha \rightarrow \gamma$.

$\alpha \rightarrow \beta\gamma$	given
$\beta\gamma \rightarrow \beta$	reflexivity rule
$\alpha \rightarrow \beta$	transitivity rule
$\beta\gamma \rightarrow \gamma$	reflexive rule
$\alpha \rightarrow \gamma$	transitive rule

7.10 Use Armstrong's axioms to prove the soundness of the pseudotransitivity rule.
Answer: Proof using Armstrong's axioms of the Pseudotransitivity Rule:
if $\alpha \rightarrow \beta$ and $\gamma\,\beta \rightarrow \delta$, then $\alpha\gamma \rightarrow \delta$.

$\alpha \rightarrow \beta$	given
$\alpha\gamma \rightarrow \gamma\,\beta$	augmentation rule and set union commutativity
$\gamma\,\beta \rightarrow \delta$	given
$\alpha\gamma \rightarrow \delta$	transitivity rule

7.11 Compute the closure of the following set *F* of functional dependencies for relation schema $R = (A, B, C, D, E)$.

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

List the candidate keys for *R*.
Answer: Compute the closure of the following set *F* of functional dependencies for relation schema $R = (A, B, C, D, E)$.

$$A \rightarrow BC$$
$$CD \rightarrow E$$
$$B \rightarrow D$$
$$E \rightarrow A$$

List the candidate keys for *R*.
Note: It is not reasonable to expect students to enumerate all of F^+. Some shorthand representation of the result should be acceptable as long as the nontrivial members of F^+ are found.
Starting with $A \rightarrow BC$, we can conclude: $A \rightarrow B$ and $A \rightarrow C$.

Since $A \rightarrow B$ and $B \rightarrow D$, $A \rightarrow D$	(decomposition, transitive)
Since $A \rightarrow CD$ and $CD \rightarrow E$, $A \rightarrow E$	(union, decomposition, transitive)
Since $A \rightarrow A$, we have	(reflexive)
$A \rightarrow ABCDE$ from the above steps	(union)
Since $E \rightarrow A$, $E \rightarrow ABCDE$	(transitive)
Since $CD \rightarrow E$, $CD \rightarrow ABCDE$	(transitive)
Since $B \rightarrow D$ and $BC \rightarrow CD$, $BC \rightarrow ABCDE$	(augmentative, transitive)
Also, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow D$, etc.	

Therefore, any functional dependency with A, E, BC, or CD on the left hand side of the arrow is in F^+, no matter which other attributes appear in the FD. Allow * to represent any set of attributes in R, then F^+ is $BD \rightarrow B$, $BD \rightarrow D$, $C \rightarrow C$, $D \rightarrow D$, $BD \rightarrow BD$, $B \rightarrow D$, $B \rightarrow B$, $B \rightarrow BD$, and all FDs of the form $A* \rightarrow \alpha$, $BC* \rightarrow \alpha$, $CD* \rightarrow \alpha$, $E* \rightarrow \alpha$ where α is any subset of $\{A, B, C, D, E\}$. The candidate keys are A, BC, CD, and E.

7.12 Using the functional dependencies of Exercise 7.11, compute B^+.
Answer: Computing B^+ by the algorithm in Figure 7.7 we start with $result = \{B\}$. Considering FDs of the form $\beta \rightarrow \gamma$ in F, we find that the only dependencies satisfying $\beta \subseteq result$ are $B \rightarrow B$ and $B \rightarrow D$. Therefore $result = \{B, D\}$. No more dependencies in F apply now. Therefore $B^+ = \{B, D\}$

7.13 Using the functional dependencies of Exercise 7.11, compute the canonical cover F_c.
Answer: The given set of FDs F is:-

$A \rightarrow BC$
$CD \rightarrow E$
$B \rightarrow D$
$E \rightarrow A$

The left side of each FD in F is unique. Also none of the attributes in the left side or right side of any of the FDs is extraneous. Therefore the canonical cover F_c is equal to F.

7.14 Consider the algorithm in Figure 7.22 to compute α^+. Show that this algorithm is more efficient than the one presented in Figure 7.7 (Section 7.3.3) and that it computes α^+ correctly.
Answer: The algorithm is correct because:

- If A is added to $result$ then there is a proof that $\alpha \rightarrow A$. To see this, observe that $\alpha \rightarrow \alpha$ trivially so α is correctly part of $result$. If $A \notin \alpha$ is added to $result$ there must be some FD $\beta \rightarrow \gamma$ such that $A \in \gamma$ and β is already a subset of $result$. (Otherwise $fdcount$ would be nonzero and the **if** condition would be false.) A full proof can be given by induction on the depth of recursion for an execution of **addin**, but such a proof can be expected only from students with a good mathematical background.
- If $A \in \alpha^+$, then A is eventually added to $result$. We prove this by induction on the length of the proof of $\alpha \rightarrow A$ using Armstrong's axioms. First observe that if procedure **addin** is called with some argument β, all the attributes in β will be added to $result$. Also if a particular FD's *fdcount* becomes 0, all the attributes in its tail will definitely be added to $result$. The base case of the proof, $A \in \alpha \Rightarrow A \in \alpha^+$, is obviously true because the first call to **addin** has the argument α. The inductive hypotheses is that if $\alpha \rightarrow A$ can be proved in n steps or less then $A \in result$. If there is a proof in $n + 1$

```
result := ∅;
/* fdcount is an array whose ith element contains the number
   of attributes on the left side of the ith FD that are
   not yet known to be in α+ */
for i := 1 to |F| do
   begin
      let β → γ denote the ith FD;
      fdcount [i] := |β|;
   end
/* appears is an array with one entry for each attribute. The
   entry for attribute A is a list of integers. Each integer
   i on the list indicates that A appears on the left side
   of the ith FD */
for each attribute A do
   begin
      appears [A] := NIL;
      for i := 1 to |F| do
         begin
            let β → γ denote the ith FD;
            if A ∈ β then add i to appears [A];
         end
   end
addin (α);
return (result);

procedure addin (α);
for each attribute A in α do
   begin
      if A ∉ result then
         begin
            result := result ∪ {A};
            for each element i of appears[A] do
               begin
                  fdcount [i] := fdcount [i] − 1;
                  if fdcount [i] := 0 then
                     begin
                        let β → γ denote the ith FD;
                        addin (γ);
                     end
               end
         end
   end
```

Figure 7.22. An algorithm to compute α^+.

steps that $\alpha \rightarrow A$, then the last step was an application of either reflexivity, augmentation or transitivity on a fact $\alpha \rightarrow \beta$ proved in n or fewer steps. If reflexivity or augmentation was used in the $(n+1)^{st}$ step, A must have been in $result$ by the end of the n^{th} step itself. Otherwise, by the inductive hypothesis $\beta \subseteq result$. Therefore the dependency used in proving $\beta \rightarrow \gamma$, $A \in \gamma$ will have $fdcount$ set to 0 by the end of the n^{th} step. Hence A will be added to $result$.

To see that this algorithm is more efficient than the one presented in the chapter note that we scan each FD once in the main program. The resulting array $appears$ has size proportional to the size of the given FDs. The recursive calls to **addin** result in processing linear in the size of $appears$. Hence the algorithm has time complexity which is linear in the size of the given FDs. On the other hand, the algorithm given in the text has quadratic time complexity, as it may perform the loop as many times as the number of FDs, in each loop scanning all of them once.

7.15 Given the database schema $R(a, b, c)$, and a relation r on the schema R, write an SQL query to test whether the functional dependency $b \rightarrow c$ holds on relation r. Also write an SQL assertion that enforces the functional dependency. Assume that no null values are present.

Answer:

a. The query is given below. Its result is non-empty if and only if $b \rightarrow c$ does not hold on r.

```
select b
from r
group by b
having count(distinct c) > 1
```

b.

```
create assertion b-to-c check
    (not exists
        (select b
         from r
         group by b
         having count(distinct c) > 1
        )
    )
```

7.16 Show that the following decomposition of the schema R of Exercise 7.2 is not a lossless-join decomposition:

(A, B, C)
(C, D, E).

Hint: Give an example of a relation r on schema R such that

$$\Pi_{A,\,B,\,C}\,(r)\ \bowtie\ \Pi_{C,\,D,\,E}\,(r)\ \neq\ r$$

Answer: Following the hint, use the following example of r:

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_2	b_2	c_1	d_2	e_2

With $R_1\ =\ (A,\ B,\ C)$, $R_2\ =\ (C,\ D,\ E)$:

a. $\Pi_{R_1}\,(r)$ would be:

A	B	C
a_1	b_1	c_1
a_2	b_2	c_1

b. $\Pi_{R_2}\,(r)$ would be:

C	D	E
c_1	d_1	e_1
c_1	d_2	e_2

c. $\Pi_{R_1}\,(r)\ \bowtie\ \Pi_{R_2}\,(r)$ would be:

A	B	C	D	E
a_1	b_1	c_1	d_1	e_1
a_1	b_1	c_1	d_2	e_2
a_2	b_2	c_1	d_1	e_1
a_2	b_2	c_1	d_2	e_2

Clearly, $\Pi_{R_1}\,(r)\ \bowtie\ \Pi_{R_2}\,(r)\ \neq\ r$. Therefore, this is a lossy join.

7.17 Let $R_1,\ R_2, \ldots, R_n$ be a decomposition of schema U. Let $u(U)$ be a relation, and let $r_i\ =\ \Pi_{R_I}(u)$. Show that

$$u\ \subseteq\ r_1\ \bowtie\ r_2\ \bowtie\ \cdots\ \bowtie\ r_n$$

Answer: Consider some tuple t in u.

Note that $r_i = \Pi_{R_i}(u)$ implies that $t[R_i] \in r_i$, $1 \leq i \leq n$. Thus,

$$t[R_1]\ \bowtie\ t[R_2]\ \bowtie\ \ldots\ \bowtie\ t[R_n]\ \in\ r_1\ \bowtie\ r_2\ \bowtie\ \ldots\ \bowtie\ r_n$$

By the definition of natural join,

$$t[R_1]\ \bowtie\ t[R_2]\ \bowtie\ \ldots\ \bowtie\ t[R_n]\ =\ \Pi_\alpha\,(\sigma_\beta\,(t[R_1]\ \times\ t[R_2]\ \times\ \ldots\ \times\ t[R_n]))$$

where the condition β is satisfied if values of attributes with the same name in a tuple are equal and where $\alpha\ =\ U$. The cartesian product of single tuples generates one tuple. The selection process is satisfied because all attributes with

the same name must have the same value since they are projections from the same tuple. Finally, the projection clause removes duplicate attribute names.

By the definition of decomposition, $U = R_1 \cup R_2 \cup \ldots \cup R_n$, which means that all attributes of t are in $t[R_1] \bowtie t[R_2] \bowtie \ldots \bowtie t[R_n]$. That is, t is equal to the result of this join.

Since t is any arbitrary tuple in u,

$$u \subseteq r_1 \bowtie r_2 \bowtie \ldots \bowtie r_n$$

7.18 Show that the decomposition in Exercise 7.2 is not a dependency-preserving decomposition.

Answer: The dependency $B \rightarrow D$ is not preserved. F_1, the restriction of F to (A, B, C) is $A \rightarrow ABC$, $A \rightarrow AB$, $A \rightarrow AC$, $A \rightarrow BC$, $A \rightarrow B$, $A \rightarrow C$, $A \rightarrow A$, $B \rightarrow B$, $C \rightarrow C$, $AB \rightarrow AC$, $AB \rightarrow ABC$, $AB \rightarrow BC$, $AB \rightarrow AB$, $AB \rightarrow A$, $AB \rightarrow B$, $AB \rightarrow C$, AC (same as AB), BC (same as AB), ABC (same as AB). F_2, the restriction of F to (C, D, E) is $A \rightarrow ADE$, $A \rightarrow AD$, $A \rightarrow AE$, $A \rightarrow DE$, $A \rightarrow A$, $A \rightarrow D$, $A \rightarrow E$, $D \rightarrow D$, E (same as A), AD, AE, DE, ADE (same as A). $(F_1 \cup F_2)^+$ is easily seen not to contain $B \rightarrow D$ since the only FD in $F_1 \cup F_2$ with B as the left side is $B \rightarrow B$, a trivial FD. We shall see in Exercise 7.22 that $B \rightarrow D$ is indeed in F^+. Thus $B \rightarrow D$ is not preserved. Note that $CD \rightarrow ABCDE$ is also not preserved.

A simpler argument is as follows: F_1 contains no dependencies with D on the right side of the arrow. F_2 contains no dependencies with B on the left side of the arrow. Therefore for $B \rightarrow D$ to be preserved there must be an FD $B \rightarrow \alpha$ in F_1^+ and $\alpha \rightarrow D$ in F_2^+ (so $B \rightarrow D$ would follow by transitivity). Since the intersection of the two schemes is A, $\alpha = A$. Observe that $B \rightarrow A$ is not in F_1^+ since $B^+ = BD$.

7.19 Show that it is possible to ensure that a dependency-preserving decomposition into 3NF is a lossless-join decomposition by guaranteeing that at least one schema contains a candidate key for the schema being decomposed. (*Hint*: Show that the join of all the projections onto the schemas of the decomposition cannot have more tuples than the original relation.)

Answer: Let F be a set of functional dependencies that hold on a schema R. Let $\sigma = \{R_1, R_2, \ldots, R_n\}$ be a dependency-preserving 3NF decomposition of R. Let X be a candidate key for R.

Consider a legal instance r of R. Let $j = \Pi_X(r) \bowtie \Pi_{R_1}(r) \bowtie \Pi_{R_2}(r) \ldots \bowtie \Pi_{R_n}(r)$. We want to prove that $r = j$.

We claim that if t_1 and t_2 are two tuples in j such that $t_1[X] = t_2[X]$, then $t_1 = t_2$. To prove this claim, we use the following inductive argument –
Let $F' = F_1 \cup F_2 \cup \ldots \cup F_n$, where each F_i is the restriction of F to the schema R_i in σ. Consider the use of the algorithm given in Figure 7.7 to compute the closure of X under F'. We use induction on the number of times that the for loop in this algorithm is executed.

- *Basis* : In the first step of the algorithm, $result$ is assigned to X, and hence given that $t_1[X] = t_2[X]$, we know that $t_1[result] = t_2[result]$ is true.

- *Induction Step* : Let $t_1[result] = t_2[result]$ be true at the end of the k th execution of the for loop.
  Suppose the functional dependency considered in the $k + 1$ th execution of the for loop is $\beta \rightarrow \gamma$, and that $\beta \subseteq result$. $\beta \subseteq result$ implies that $t_1[\beta] = t_2[\beta]$ is true. The facts that $\beta \rightarrow \gamma$ holds for some attribute set R_i in σ, and that $t_1[R_i]$ and $t_2[R_i]$ are in $\Pi_{R_i}(r)$ imply that $t_1[\gamma] = t_2[\gamma]$ is also true. Since γ is now added to $result$ by the algorithm, we know that $t_1[result] = t_2[result]$ is true at the end of the $k + 1$ th execution of the for loop.

Since σ is dependency-preserving and X is a key for R, all attributes in R are in $result$ when the algorithm terminates. Thus, $t_1[R] = t_2[R]$ is true, that is, $t_1 = t_2$ – as claimed earlier.

Our claim implies that the size of $\Pi_X(j)$ is equal to the size of j. Note also that $\Pi_X(j) = \Pi_X(r) = r$ (since X is a key for R). Thus we have proved that the size of j equals that of r. Using the result of Exercise 7.17, we know that $r \subseteq j$. Hence we conclude that $r = j$.

Note that since X is trivially in 3NF, $\sigma \cup \{X\}$ is a dependency-preserving lossless-join decomposition into 3NF.

7.20 List the three design goals for relational databases, and explain why each is desirable.
Answer: The three design goals are lossless-join decompositions, dependency preserving decompositions, and minimization of repetition of information. They are desirable so we can maintain an accurate database, check correctness of updates quickly, and use the smallest amount of space possible.

7.21 Give a lossless-join decomposition into BCNF of schema *R* of Exercise 7.2.
Answer: From Exercise 7.11, we know that $B \rightarrow D$ is nontrivial and the left hand side is not a superkey. By the algorithm of Figure 7.13 we derive the relations $\{(A, B, C, E), (B, D)\}$. This is in BCNF.

7.22 Give an example of a relation schema R' and set F' of functional dependencies such that there are at least three distinct lossless-join decompositions of R' into BCNF.
Answer: Given the relation $R' = (A, B, C, D)$ the set of functional dependencies $F' = A \rightarrow B, C \rightarrow D, B \rightarrow C$ allows three distinct BCNF decompositions.

$$R_1 = \{(A, B), (C, D), (B, C)\}$$

is in BCNF as is

$$R_2 = \{(A, B), (C, D), (A, C)\}$$

$$R_2 = \{(A, B), (C, D), (A, C)\}$$

$$R_3 = \{(B, C), (A, D), (A, B)\}$$

7.23 In designing a relational database, why might we choose a non-BCNF design?
Answer: BCNF is not always dependency preserving. Therefore, we may want to choose another normal form (specifically, 3NF) in order to make checking dependencies easier during updates. This would avoid joins to check dependencies and increase system performance.

7.24 Give a lossless-join, dependency-preserving decomposition into 3NF of schema R of Exercise 7.2.
Answer: First we note that the dependencies given in Exercise 7.2 form a canonical cover. Generating the schema from the algorithm of Figure 7.14 we get

$$R' = \{(A, B, C), (C, D, E), (B, D), (E, A)\}.$$

Schema $(A,\ B,\ C)$ contains a candidate key. Therefore R' is a third normal form dependency-preserving lossless-join decomposition.

Note that the original schema $R = (A,\ B,\ C,\ D,\ E)$ is already in 3NF. Thus, it was not necessary to apply the algorithm as we have done above. The single original schema is trivially a lossless join, dependency-preserving decomposition.

7.25 Let a *prime* attribute be one that appears in at least one candidate key. Let α and β be sets of attributes such that $\alpha \rightarrow \beta$ holds, but $\beta \rightarrow \alpha$ does not hold. Let A be an attribute that is not in α, is not in β, and for which $\beta \rightarrow A$ holds. We say that A is *transitively dependent* on α. We can restate our definition of 3NF as follows: A relation schema R is in 3NF with respect to a set F of functional dependencies if there are no nonprime attributes A in R for which A is transitively dependent on a key for R.

Show that this new definition is equivalent to the original one.
Answer: Suppose R is in 3NF according to the textbook definition. We show that it is in 3NF according to the definition in the exercise. Let A be a nonprime attribute in R that is transitively dependent on a key α for R. Then there exists $\beta \subseteq R$ such that $\beta \rightarrow A$, $\alpha \rightarrow \beta$, $A \notin \alpha$, $A \notin \beta$, and $\beta \rightarrow \alpha$ does not hold. But then $\beta \rightarrow A$ violates the textbook definition of 3NF since

- $A \notin \beta$ implies $\beta \rightarrow A$ is nontrivial
- Since $\beta \rightarrow \alpha$ does not hold, β is not a superkey
- A is not any candidate key, since A is nonprime

Now we show that if R is in 3NF according to the exercise definition, it is in 3NF according to the textbook definition. Suppose R is not in 3NF according the the textbook definition. Then there is an FD $\alpha \rightarrow \beta$ that fails all three conditions. Thus

- $\alpha \rightarrow \beta$ is nontrivial.
- α is not a superkey for R.
- Some A in $\beta - \alpha$ is not in any candidate key.

This implies that A is nonprime and $\alpha \rightarrow A$. Let γ be a candidate key for R. Then $\gamma \rightarrow \alpha$, $\alpha \rightarrow \gamma$ does not hold (since α is not a superkey), $A \notin \alpha$, and

$A \notin \gamma$ (since A is nonprime). Thus A is transitively dependent on γ, violating the exercise definition.

7.26 A functional dependency $\alpha \rightarrow \beta$ is called a **partial dependency** if there is a proper subset γ of α such that $\gamma \rightarrow \beta$. We say that β is *partially dependent* on α. A relation schema R is in **second normal form** (2NF) if each attribute A in R meets one of the following criteria:

- It appears in a candidate key.
- It is not partially dependent on a candidate key.

Show that every 3NF schema is in 2NF. (*Hint*: Show that every partial dependency is a transitive dependency.)

Answer: Referring to the definitions in Exercise 7.25, a relation schema R is said to be in 3NF if there is no non-prime attribute A in R for which A is transitively dependent on a key for R.

We can also rewrite the definition of 2NF given here as :
"A relation schema R is in 2NF if no non-prime attribute A is partially dependent on any candidate key for R."

To prove that every 3NF schema is in 2NF, it suffices to show that if a non-prime attribute A is partially dependent on a candidate key α, then A is also transitively dependent on the key α.

Let A be a non-prime attribute in R. Let α be a candidate key for R. Suppose A is partially dependent on α.

- From the definition of a partial dependency, we know that for some proper subset β of α, $\beta \rightarrow A$.
- Since $\beta \subset \alpha$, $\alpha \rightarrow \beta$. Also, $\beta \rightarrow \alpha$ does not hold, since α is a candidate key.
- Finally, since A is non-prime, it cannot be in either β or α.

Thus we conclude that $\alpha \rightarrow A$ is a transitive dependency. Hence we have proved that every 3NF schema is also in 2NF.

7.27 Given the three goals of relational-database design, is there any reason to design a database schema that is in 2NF, but is in no higher-order normal form? (See Exercise 7.26 for the definition of 2NF.)

Answer: The three design goals of relational databases are to avoid

- Repetition of information
- Inability to represent information
- Loss of information.

2NF does not prohibit as much repetition of information since the schema (A, B, C) with dependencies $A \rightarrow B$ and $B \rightarrow C$ is allowed under 2NF, although the same (B, C) pair could be associated with many A values, needlessly duplicating C values. To avoid this we must go to 3NF. Repetition of information is allowed in 3NF in some but not all of the cases where it is allowed in 2NF. Thus, in general, 3NF reduces repetition of information. Since we can always achieve a lossless join 3NF decomposition, there is no loss of information needed in going from 2NF to 3NF.

Note that the decomposition $\{(A, B), (B, C)\}$ is a dependency-preserving and lossless-loin 3NF decomposition of the schema (A, B, C). However, in case we choose this decomposition, retrieving information about the relationship between A, B and C requires a join of two relations, which is avoided in the corresponding 2NF decomposition.

Thus, the decision of which normal form to choose depends upon how the cost of dependency checking compares with the cost of the joins. Usually, the 3NF would be preferred. Dependency checks need to be made with *every* insert or update to the instances of a 2NF schema, whereas, only some queries will require the join of instances of a 3NF schema.

7.28 Give an example of a relation schema R and a set of dependencies such that R is in BCNF, but is not in 4NF.
Answer: The relation schema $R = (A, B, C, D, E)$ and the set of dependencies

$$\begin{aligned} A &\rightarrow\!\!\!\!\rightarrow BC \\ B &\rightarrow\!\!\!\!\rightarrow CD \\ E &\rightarrow\!\!\!\!\rightarrow AD \end{aligned}$$

constitute a BCNF decomposition, however it is clearly not in 4NF. (It is BCNF because all FDs are trivial).

7.29 Explain why 4NF is a normal form more desirable than BCNF.
Answer: 4NF is more desirable than BCNF because it reduces the repetition of information. If we consider a BCNF schema not in 4NF (see Exercise 7.28), we observe that decomposition into 4NF does not lose information provided that a lossless join decomposition is used, yet redundancy is reduced.

7.30 Explain how dangling tuples may arise. Explain problems that they may cause.
Answer: Dangling tuples can arise when one tuple is inserted into a decomposed relation but no corresponding tuple is inserted into the other relations in the decomposition. They can cause incorrect values to be returned by queries which form the join of a decomposed relation since the dangling tuple might not be included. As we saw in Chapter 5, dangling tuples can be avoided by the specification of referential integrity constraints.

CHAPTER 8

Object-Oriented Databases

This chapter provides an introduction to object-oriented databases. This chapter and the next chapter form a logical unit and should be taught consecutively. It is possible to teach these chapters before covering normalization (Chapter 7).

The sections of the chapter prior to the section on persistent C++ and ODMG (Section 8.5) do not assume any familiarity with an object-oriented programming language. However, it is quite possible that students may already be familiar with the basic concepts of object orientation, and with an object-oriented programming languages. For such students Section 8.2 can be covered relatively quickly. However, it is important to point out the motivation for object-oriented features in the context of a database, and how the requirements differ from those of a programming language.

There is a tendency to confuse "persistent" object-oriented languages with object-oriented databases. A persistent object-oriented language should be merely a front-end to a database. It is important to remind students of all of the features that a database system must have, so that, they can distinguish full-fledged object-oriented database systems from systems that provide an object-oriented front-end, but provide little in the way of database facilities such as a query facility, an on-line catalog, concurrency control and recovery.

There are several commercial object-oriented database systems available on the market, and a few public domain systems as well. Some of the commercial systems also offer low-cost or free copies for academic use. The commercial object-oriented database systems include Objectivity (www.objectivity.com), ObjectStore (www.odi.com), and Versant (www.versant.com).

Changes from 3rd edition:
Some examples have been updated to make them more intuitive. The coverage of ODMG has been updated to ODMG-2, including the new syntax (with a d_ prefix for keywords), and the new d_rel_ref feature to declare relationships.

Exercises

8.1 For each of the following application areas, explain why a relational database system would be inadequate. List all specific system components that would need to be modified.

a. Computer-aided design

b. Multimedia databases

Answer: Each of the applications includes large, specialized data items (e.g., a program module, a graphic image, digitized voice, a document). These data items have operations specific to them (e.g., compile, rotate, play, format) that cannot be expressed in relational query languages. These data items are of variable length making it impractical to store them in the short fields that are allowed in records for such database systems. Thus, the data model, data manipulation language, and data definition language need to be changed.

Also, long-duration and nested transactions are typical of these applications. Changes to the concurrency and recovery subsystems are likely to be needed.

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

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

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

Construct an object-oriented database schema definition for this database. Use inheritance where appropriate.

Answer:

```
class vehicle {
    int         vehicle-id;
    string      license-number;
    string      manufacturer;
    string      model;
    date        purchase-date;
```

```
    string      color;
};

class truck isa vehicle {
    int         cargo-capacity;
};

class sports-car isa vehicle {
    int         horsepower;
    int         renter-age-requirement;
};

class van isa vehicle {
    int         num-passengers;
};

class off-road-vehicle isa vehicle {
    real        ground-clearance;
    string      drivetrain;
};
```

8.4 Explain why ambiguity potentially exists with multiple inheritance. Illustrate your explanation with an example.

Answer: A class inherits the variables and methods of all its immediate super-classes. Thus it could inherit a variable or method of the same name from more than one super-class. When that particular variable or method of an object of the sub-class is referenced, there is an ambiguity regarding which of the super-classes provides the inheritance.

For instance, let there be classes *teacher* and *student*, both having a variable *department*. If a class *teachingAssistant* inherits from both of these classes, any reference to the *department* variable of a *teachingAssistant* object is ambiguous.

8.5 Explain how the concept of object identity in the object-oriented model differs from the concept of tuple equality in the relational model.

Answer: Tuple equality is determined by data values. Object identity is independent of data values, since object-oriented systems use built-in identity.

8.6 Explain the distinction in meaning between edges in a DAG representing inheritance and a DAG representing object containment.

Answer: An edge from class A to class B in the DAG representing inheritance means that an object of class B is also an object of class A. It has all the properties that objects of class A have, plus additional ones of its own. In particular, it inherits all the variables and methods of class A. It can of course provide its own implementations for the inherited methods.

And edge from class *A* to class *B* in the object containment DAG means that an object of class *A* contains an object of class *B*. There need not be any similarities in the properties of *A* and *B*. Neither *B* nor *A* inherit anything from the other. They function as independent types, to the extent that an object of class *A* can access the variables of the *B* object contained in it only via the *B* object's methods.

8.7 Why do persistent programming languages allow transient objects? Might it be simpler to use only persistent objects, with unneeded objects deleted at the end of an execution? Explain your answer.

Answer: Creation, destruction and access will typically be more time consuming and expensive for persistent objects stored in the database, than for transient objects in the transaction's local memory. This is because of the over-heads in preserving transaction semantics, security and integrity. Since a transient object is purely local to the transaction which created it and does not enter the database, all these over-heads are avoided. Thus, in order to provide efficient access to purely local and temporary data, transient objects are provided by persistent programming languages.

8.8 Using ODMG C++

a. Give schema definitions corresponding to the relational schema shown in Figure 3.39, using references to express foreign-key relationships.

b. Write programs to compute each of the queries in Exercise 3.10.

Answer:

a. The schema definitions can be written in two different ways, one of which is a direct translation from the relational schema, while the other uses object-oriented features more directly.

- The first scheme is as follows:

```
class employee : public d_Object {
public:
    d_String person-name;
    d_String street;
    d_String city;
};

class company : public d_Object {
public:
    d_String company-name;
    d_String city;
};

class works : public d_Object {
public:
    d_Ref<employee> person;
```

```
    d_Ref<company> comp;
    d_Long salary;
};

class manages : public d_Object {
public:
    d_Ref<employee> person;
    d_Ref<employee> manager;
};
```

- The second schema is as follows

```
class employee : public d_Object {
public:
    d_String person-name;
    d_String street;
    d_String city;
    d_Rel_Ref<company, _employees> comp;
    d_Ref<employee> manager;
    d_Long salary;
};

class company : public d_Object {
public:
    d_String company-name;
    d_String city;
    d_Rel_Set<employee, _comp> employees;
};

const char _employees[] = "employees";
const char _comp[] = "comp";
```

b. We present queries for the second schema.

- Find the company with the most employees.

```
d_Ref<company> mostemployees(){
        d_Database emp_db_obj;
        d_Database * emp_db = &emp_db_obj;
        emp_db->open("Emp-DB");
        d_Transaction Trans;
        Trans.begin();
        d_Extent<company> all_comps(emp_db);
        d_Iterator<d_Ref<company>> iter=all_comps.create_iterator();
        d_Iterator<d_Ref<employee>> iter2;
        d_Ref<company> c, maxc;
        d_Ref<employee> e;
        int count;
        int maxcount=0;
        while(iter.next(c)) {
                iter2=(c->employees).create_iterator();
                count=0;
                while(iter2.next(e)) {
                        count++;
                }
                if(maxcount < count) {
                        maxcount=count;
                        maxc=c;
                }
        }
        Trans.commit();
        return maxc;
}
```

- Find the company with the smallest payroll.

```
d_Ref<company> smallestpay(){
        d_Database emp_db_obj;
        d_Database * emp_db = &emp_db_obj;
        emp_db->open("Emp-DB");
        d_Transaction Trans;
        Trans.begin();
        d_Extent<company> all_comps(emp_db);
        d_Iterator<d_Ref<company>> iter=all_comps.create_iterator();
        d_Iterator<d_Ref<employee>> iter2;
        d_Ref<company> c, minc;
        d_Ref<employee> e;
        d_Long sal;
        d_Long minsal=0;
        while(iter.next(c)) {
                iter2=(c->employees).create_iterator();
                sal=0;
                while(iter2.next(e)) {
                        sal+=e->salary;
                }
                if(minsal > sal) {
                        minsal=sal;
                        minc=c;
                }
        }
        Trans.commit();
        return minc;
}
```

- Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

```
d_Set<d_Ref<company>> highersal(){
    d_Database emp_db_obj;
    d_Database * emp_db = &emp_db_obj;
    emp_db->open("Emp-DB");
    d_Transaction Trans;
    Trans.begin();
    d_Extent<company> all_comps(emp_db);
    d_Iterator<d_Ref<company>> iter=all_comps.create_iterator();
    d_Iterator<d_Ref<employee>> iter2;
    d_Ref<company> c, FBC=all_comps.select(
            "company-name='First Bank Corporation'");
    d_Set<d_Ref<company>> result;
    d_Ref<employee> e;
    int count;
    d_Long avsal=0, avFBCsal=0, sal=0;
    iter2=(FBC->employees).create_iterator();
    while(iter2.next(e)) {
        count++;
        sal+=e->salary;
    }
    avFBCsal=sal/count;
    while(iter.next(c)) {
        iter2=(c->employees).create_iterator();
        sal=0; count=0;
        while(iter2.next(e)) {
            sal+=e->salary;
            count++;
        }
        avsal=sal/count;
        if(avsal > avFBCsal) {
            result.insert_element(c);
        }
    }
    Trans.commit();
    return result;
}
```

8.9 Using ODMG C++, give schema definitions corresponding to the E-R diagram in Figure 2.29, using references to implement relationships.
Answer:

```
class person : public d_Object {
public:
    d_String name;
    d_String address;
    d_String phone;
};
```

```
class author : public person {
public:
    d_String URL;
    d_Rel_Set<book, _authors> books;
};

class publisher : public person {
public:
    d_String URL;
    d_Rel_Set<book, _book_publisher> books;
};

class customer : public person {
public:
    d_String email;
    d_Rel_Set<shoppingbasket, _owner> baskets;
};

class book : public d_Object {
public:
    int year;
    d_String title;
    float price;
    d_String ISBN;
    d_Rel_Set<author, _books> authors;
    d_Rel_Ref<publisher, _books> book_publisher;
};

class shoppingbasket : public d_Object {
public:
    d_String basketID;
    d_Rel_Ref<customer, _baskets> owner;
    d_Set<book_qty> contains;
};

class warehouse : public d_Object {
public:
    d_String address;
    d_String phone;
    d_String code;
    d_Set<book_qty> stocks;
};

class book_qty : public d_Object {
public:
```

```
    d_Ref<book> book;
    int number;
};

const char _books[] = "books";
const char _authors[] = "authors";
const char _book_publisher[] = "book_publisher";
const char _baskets[] = "baskets";
const char _owner[] = "owner";
```

8.10 Explain, using an example, how to represent a ternary relationship in an object-oriented data model such as ODMG C++.

Answer: To represent ternary relationships, create a class corresponding to the relationship and refer to the entities in this class. For example, to represent the ternary relationship in Figure 2.13, we do the following:

```
class workson : public d_Object {
public:
    d_Ref<employee> emp;
    d_Ref<branch> branch;
    d_Ref<job> job;
};
```

8.11 Explain how a persistent pointer is implemented. Contrast this implementation with that of pointers as they exist in general-purpose languages, such as C or Pascal.

Answer: Persistent pointers can be implemented as Abstract Data Types (ADTs). These ADTs should provide the typical pointer operations like incrementing and dereferencing, so their usage and regular pointer usage is uniform. Regular pointers on the other hand are usually built-in types, implemented as part of the language.

8.12 If an object is created without any references to it, how can that object be deleted?

Answer: If an object is created without any references to it, it can neither be accessed nor deleted via a program. The only way is for the database system to locate and delete such objects by itself. This is called *garbage collection*. One way to do garbage collection is by the method of *mark and sweep*. First, the objects referred to directly by programs are marked. Then references from these objects to other objects are followed, and those referred objects are marked. This procedure is followed repeatedly until no more unmarked objects can be reached by following reference chains from the marked objects. At this point, all these remaining unmarked objects are deleted. This method is correct; we can prove

that if no new objects are marked after a round of mark and sweep, the remaining unmarked objects are indeed unreferenced.

8.13 Consider a system that provides persistent objects. Is such a system necessarily a database system? Explain your answer.

Answer: A database system must provide for such features as transactions, queries (associative retrieval of objects), security, and integrity. A persistent object system may not offer such features.

CHAPTER 9

Object-Relational Databases

This chapter describes extensions to relational database systems to provide complex data types and object-oriented features. Such extended systems are called object-relational systems. Since the chapter was introduced in the 3^{rd} edition most commercial database systems have added some support for object-relational features, and these features have been standardized as part of SQL:1999.

It would be instructive to assign students exercises aimed at finding applications where the object-relational model, in particular complex objects, would be better suited than the traditional relational model.

Changes from 3^{rd} edition:

The query language features are now based on the SQL:1999 standard, which was not ready when the 3^{rd} edition was published; that edition was based from features from several different proposals for extending SQL.

Exercises

9.1 Consider the database schema

Emp = (ename, **setof***(Children),* **setof***(Skills))*
Children = (name, Birthday)
Birthday = (day, month, year)
Skills = (type, **setof***(Exams))*
Exams = (year, city)

Assume that attributes of type **setof***(Children)*, **setof***(Skills)*, and **setof***(Exams)*, have attribute names *ChildrenSet*, *SkillsSet*, and *ExamsSet*, respectively. Suppose the database contains a relation *emp (Emp)*. Write the following queries in SQL:1999 (with the extensions described in this chapter).

a. Find the names of all employees who have a child who has a birthday in March.
b. Find those employees who took an examination for the skill type "typing" in the city "Dayton".
c. List all skill types in the relation *emp*.

Answer:

a. Find the names of all employees who have a child who has a birthday in March.

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

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

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

c. List all skill types in the relation *emp*.

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

9.2 Redesign the database of Exercise 9.1 into first normal form and fourth normal form. List any functional or multivalued dependencies that you assume. Also list all referential-integrity constraints that should be present in the first- and fourth-normal-form schemas.
Answer: To put the schema into first normal form, we flatten all the attributes into a single relation schema.

Employee-details = (*ename, cname, bday, bmonth, byear, stype, xyear, xcity*)

We rename the attributes for the sake of clarity. *cname* is *Children.name*, and *bday, bmonth, byear* are the *Birthday* attributes. *stype* is *Skills.type*, and *xyear* and *xcity* are the *Exams* attributes. The FDs and multivalued dependencies we assume are:-

$$\begin{aligned} ename, cname &\rightarrow bday, bmonth, byear \\ ename &\twoheadrightarrow cname, bday, bmonth, byear \\ ename, stype &\twoheadrightarrow xyear, xcity \end{aligned}$$

The FD captures the fact that a child has a unique birthday, under the assumption that one employee cannot have two children of the same name. The MVDs capture the fact there is no relationship between the children of an employee and his or her skills-information.

The redesigned schema in fourth normal form is:-

Employee = (*ename*)
Child = (*ename, cname, bday, bmonth, byear*)
Skill = (*ename, stype, xyear, xcity*)

ename will be the primary key of *Employee*, and (*ename, cname*) will be the primary key of *Child*. The *ename* attribute is a foreign key in *Child* and in *Skill*, referring to the *Employee* relation.

9.3 Consider the schemas for the table *people*, and the tables *students* and *teachers*, which were created under *people*, in Section 9.3. Give a relational schema in third normal form that represents the same information. Recall the constraints on subtables, and give all constraints that must be imposed on the relational schema so that every database instance of the relational schema can also be represented by an instance of the schema with inheritance.
Answer: A corresponding relational schema in third normal form is given below:-

People = (*name, address*)
Students = (*name, degree, student-department*)
Teachers = (*name, salary, teacher-department*)

name is the primary key for all the three relations, and it is also a foreign key referring to *People*, for both *Students* and *Teachers*.

Instead of placing only the *name* attribute of *People* in *Students* and *Teachers*, both its attributes can be included. In that case, there will be a slight change, namely – (*name, address*) will become the foreign key in *Students* and *Teachers*. The primary keys will remain the same in all tables.

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

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

Construct an SQL:1999 schema definition for this database. Use inheritance where appropriate.
Answer: For this problem, we use table inheritance. We assume that **MyDate**, **Color** and **DriveTrainType** are pre-defined types.

create type *Vehicle*
(*vehicle-id* **integer**,
license-number **char(15)**,
manufacturer **char(30)**,

```
    model char(30),
    purchase-date MyDate,
    color Color)

create table vehicle of type Vehicle

create table truck
    (cargo-capacity integer)
    under vehicle

create table sportsCar
    (horsepower integer
     renter-age-requirement integer)
    under vehicle

create table van
    (num-passengers integer)
    under vehicle

create table offRoadVehicle
    (ground-clearance real
     driveTrain DriveTrainType)
    under vehicle
```

9.5 Explain the distinction between a type x and a reference type **ref**(x). Under what circumstances would you choose to use a reference type?
Answer: If the type of an attribute is x, then in each tuple of the table, corresponding to that attribute, there is an actual object of type x . If its type is **ref**(x), then in each tuple, corresponding to that attribute, there is a *reference* to some object of type x. We choose a reference type for an attribute, if that attribute's intended purpose is to refer to an independent object.

9.6 Consider the E-R diagram in Figure 2.11, which contains composite, multivalued and derived attributes.

a. Give an SQL:1999 schema definition corresponding to the E-R diagram. Use an array to represent the multivalued attribute, and appropriate SQL:1999 constructs to represent the other attribute types.

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

Answer:

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

```
create type Name
```

```
    (first-name varchar(15),
     middle-initial char,
     last-name varchar(15))
create type Street
    (street-name varchar(15),
     street-number varchar(4),
     apartment-number varchar(7))
create type Address
    (street Street,
     city varchar(15),
     state varchar(15),
     zip-code char(6))
create table customer
    (name Name,
     customer-id varchar(10),
     address Adress,
     phones  char(7)  array[10],
     dob date)
method integer age()
```

b.

```
create function Name (f varchar(15), m char, l varchar(15))
returns Name
begin
    set first-name = f;
    set middle-initial = m;
    set last-name = l;
end
create function Street (sname varchar(15), sno varchar(4), ano varchar(7))
returns Street
begin
    set street-name = sname;
    set street-number = sno;
    set apartment-number =ano;
end
create function Address (s Street, c varchar(15), sta varchar(15), zip varchar(6))
returns Address
begin
    set street = s;
    set city = c;
    set state =sta;
    set zip-code =zip;
end
```

9.7 Give an SQL:1999 schema definition of the E-R diagram in Figure 2.17, which contains specializations.

Answer:

```
create type Person
    (name varchar(30),
     street varchar(15),
     city varchar(15))
create type Employee
    under Person
    (salary integer)
create type Customer
    under Person
    (credit-rating integer)
create type Officer
    under Employee
    (office-number integer)
create type Teller
    under Employee
    (station-number integer,
    hours-worked integer)
create type Secretary
    under Employee
    (hours-worked integer)
create table person of Person
create table employee of Employee
    under person
create table customer of Customer
    under person
create table officer of Officer
    under employee
create table teller of Teller
    under employee
create table secretary of Secretary
    under employee
```

9.8 Consider the relational schema shown in Figure 3.39.

a. Give a schema definition in SQL:1999 corresponding to the relational schema, but using references to express foreign-key relationships.

b. Write each of the queries in Exercise 3.10 on the above schema, using SQL:1999.

Answer:

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

```
create type Employee
    (person-name varchar(30),
     street varchar(15),
```

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

b. i.
```
select comp->name
from works
group by comp
having count(person) ≥ all(select count(person)
                           from works
                           group by comp)
```

ii.
```
select comp->name
from works
group by comp
having sum(salary) ≤ all(select sum(salary)
                         from works
                         group by comp)
```

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

9.9 Consider an employee database with two relations

employee (*employee-name*, *street*, *city*)
works (*employee-name*, *company-name*, *salary*)

where the primary keys are underlined. Write a query to find companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

a. Using SQL:1999 functions as appropriate.
b. Without using SQL:1999 functions.

Answer:

a.
```
create function avg-salary(cname varchar(15))
    returns integer
    declare result integer;
        select avg(salary) into result
        from works
        where works.company-name = cname
    return result;
end
select company-name
from works
where avg-salary(company-name) > avg-salary("First Bank Corporation")
```

b.
```
select company-name
from works
group by company-name
having avg(salary) > (select avg(salary)
                      from works
                      where company-name="First Bank Corporation")
```

9.10 Rewrite the query in Section 9.6.1 that returns the titles of all books that have more than one author, using the **with** clause in place of the function.

Answer:

```
with multauthors(title, count) as
    select title, count(author)
    from authors
    group by title
select books4.title
from books4, multauthors
where books4.title = multauthors.title
and multauthors.count > 1
```

9.11 Compare the use of embedded SQL with the use in SQL of functions defined in a general-purpose programming language. Under what circumstances would you use each of these features?

Answer: SQL functions are primarily a mechanism for extending the power of SQL to handle attributes of complex data types (like images), or to perform complex and non-standard operations. Embedded SQL is useful when imperative actions like displaying results and interacting with the user are needed. These cannot be done conveniently in an SQL only environment. Embedded SQL can be used instead of SQL functions by retrieving data and then performing the function's operations on the SQL result. However a drawback is that a lot of query-evaluation functionality may end up getting repeated in the host language code.

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

a. A computer-aided design system for a manufacturer of airplanes
b. A system to track contributions made to candidates for public office
c. An information system to support the making of movies

Answer:

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

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

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

CHAPTER 10

XML

In the 4 1/2 years since the previous edition was published, XML has gone from a little known proposal to the World Wide Web Consortium, to an extensive set of standards that are being used widely, and whose use is growing rapidly. In this period the goals of XML have changed from being a better form SGML or HTML, into becoming the primary data model for data interchange.

Our view of XML is decidedly database centric: it is important to be aware that many uses of XML are document centric, but we believe the bulk of XML applications will be in data representation and interchange between database applications. In this view, XML is a data model that provides a number of features beyond that provided by the relational model, in particular the ability to package related information into a single unit, by using nested structures. Specific application domains for data representation and interchange need their own standards that define the data schema.

Given the extensive nature of XML and related standards, this chapter only attempts to provide an introduction, and does not attempt to provide a complete description. For a course that intends to explore XML in detail, supplementary material may be required. These could include online information on XML and books on XML.

Exercises

10.1 Give an alternative representation of bank information containing the same data as in Figure 10.1, but using attributes instead of subelements. Also give the DTD for this representation.

Answer:

a. XML representation of data using attributes:

```
<bank>
    <account account-number="A-101" branch-name="Downtown"
        balance="500">
    </account>
    <account account-number="A-102" branch-name="Perryridge"
        balance="400">
    </account>
    <account account-number="A-201" branch-name="Brighton"
        balance="900">
    </account>
    <customer customer-name="Johnson" customer-street="Alma"
        customer-city="Palo Alto">
    </customer>
    <customer customer-name="Hayes" customer-street="Main"
        customer-city="Harrison">
    </customer>
    <depositor account-number="A-101" customer-name="Johnson">
    </depositor>
    <depositor account-number="A-201" customer-name="Johnson">
    </depositor>
    <depositor account-number="A-102" customer-name="Hayes">
    </depositor>
</bank>
```

b. DTD for the bank:

```
<!DOCTYPE bank [
    <!ELEMENT account >
    <!ATTLIST account
        account-number ID #REQUIRED
        branch-name CDATA #REQUIRED
        balance CDATA #REQUIRED >
    <!ELEMENT customer >
    <!ATTLIST customer
        customer-name ID #REQUIRED
        customer-street CDATA #REQUIRED
        customer-street CDATA #REQUIRED >
    <!ELEMENT depositor >
    <!ATTLIST depositor
        account-number IDREF #REQUIRED
        customer-name IDREF #REQUIRED >
] >
```

10.2 Show, by giving a DTD, how to represent the *books* nested-relation from Section 9.1, using XML.

Answer:

```
<!DOCTYPE bib [
    <!ELEMENT book (title, author+, publisher, keyword+)>
    <!ELEMENT publisher (pub-name, pub-branch) >
    <!ELEMENT title ( #PCDATA )>
    <!ELEMENT author ( #PCDATA )>
    <!ELEMENT keyword ( #PCDATA )>
    <!ELEMENT pub-name( #PCDATA )>
    <!ELEMENT pub-branch( #PCDATA )>
] >
```

10.3 Give the DTD for an XML representation of the following nested-relational schema

Emp = (ename, ChildrenSet **setof***(Children), SkillsSet* **setof***(Skills))*
Children = (name, Birthday)
Birthday = (day, month, year)
Skills = (type, ExamsSet **setof***(Exams))*
Exams = (year, city)

Answer:

```
<!DOCTYPE db [
    <!ELEMENT emp (ename, children*, skills*)>
    <!ELEMENT children (name, birthday)>
    <!ELEMENT birthday (day, month, year)>
    <!ELEMENT skills (type, exams+)>
    <!ELEMENT exams (year, city)>
    <!ELEMENT ename( #PCDATA )>
    <!ELEMENT name( #PCDATA )>
    <!ELEMENT day( #PCDATA )>
    <!ELEMENT month( #PCDATA )>
    <!ELEMENT year( #PCDATA )>
    <!ELEMENT type( #PCDATA )>
    <!ELEMENT city( #PCDATA )>
] >
```

10.4 Write the following queries in XQuery, assuming the DTD from Exercise 10.3.

a. Find the names of all employees who have a child who has a birthday in March.
b. Find those employees who took an examination for the skill type "typing" in the city "Dayton".
c. List all skill types in *Emp*.

Answer:

a. Find the names of all employees who have a child who has a birthday in March.

```
for $e in /db/emp,
    $m in distinct($e/children/birthday/month)
where $m = 'March'
return $e/ename
```

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

```
for $e in /db/emp
    $s in $e/skills[type='typing']
    $exam in $s/exams
where $exam/city= 'Dayton'
return $e/ename
```

c. List all skill types in *Emp*.

```
for $t in distinct (/db/emp/skills/type)
return $t
```

10.5 Write queries in XSLT and in XPath on the DTD of Exercise 10.3 to list all skill types in *Emp*.
Answer:

a. XPath: /db/emp/skills/type

b. XSLT:

```
<xsl:template match="/db/emp">
    <skills>
    <xsl:apply-templates/>
    </skills>
</xsl:template>
<xsl:template match="/skills">
    <type>
    <xsl:value-of select="type"/>
    </type>
</xsl:template>
<xsl:template match="."/>
```

10.6 Write a query in XQuery on the XML representation in Figure 10.1 to find the total balance, across all accounts, at each branch. (Hint: Use a nested query to get the effect of an SQL **group by**.)
Answer:

```
for $b in distinct (/bank/account/branch-name)
return
<branch-total>
    <branch-name> $b/text() </branch-name>
    let $s := sum (/bank/account[branch-name=$b]/balance)
    return <total-balance> $s </total-balance>
</branch-total>
```

10.7 Write a query in XQuery on the XML representation in Figure 10.1 to compute the left outer join of customer elements with account elements. (Hint: Use universal quantification.)
Answer:

```
<lojoin>
for $b in /bank/account,
      $c in /bank/customer,
      $d in /bank/depositor
where $a/account-number = $d/account-number
      and $c/customer-name = $d/customer-name
return <cust-acct> $c $a </cust-acct>
|
for $c in /bank/customer,
where every $d in /bank/depositor satisfies
(not ($c/customer-name=$d/customer-name))
return <cust-acct> $c </cust-acct>
</lojoin>
```

10.8 Give a query in XQuery to flip the nesting of data from Exercise 10.2. That is, at the outermost level of nesting the output must have elements corresponding to authors, and each such element must have nested within it items corresponding to all the books written by the author.
Answer:

```
for $a in distinct (/bib/book/author)
return
      <author>
            <author-name> $a/text() </author-name>
            for $b in (/bib/book/[author=$a])
            return $b
      < \author>
```

10.9 Give the DTD for an XML representation of the information in Figure 2.29. Create a separate element type to represent each relationship, but use ID and IDREF to implement primary and foreign keys.
Answer: The answer is given in Figure 10.1.

10.10 Write queries in XSLT and XQuery to output customer elements with associated account elements nested within the customer elements, given the bank information representation using ID and IDREFS in Figure 10.8.
Answer: The answer in XQuery is

```
<!DOCTYPE bookstore [
    <!ELEMENT basket (contains+, basket-of)>
    <!ATTLIST basket
        basketid ID #REQUIRED >
    <!ELEMENT customer (name, address, phone)>
    <!ATTLIST customer
        email ID #REQUIRED >
    <!ELEMENT book (year, title, price, written-by, published-by)>
    <!ATTLIST book
        ISBN ID #REQUIRED >
    <!ELEMENT warehouse (address, phone, stocks)>
    <!ATTLIST warehouse
        code ID #REQUIRED >
    <!ELEMENT author (name, address, URL)>
    <!ATTLIST author
        authid ID #REQUIRED >
    <!ELEMENT publisher (address, phone)>
    <!ATTLIST publisher
        name ID #REQUIRED >
    <!ELEMENT basket-of >
    <!ATTLIST basket-of
        owner IDREF #REQUIRED >
    <!ELEMENT contains >
    <!ATTLIST contains
        book IDREF #REQUIRED
        number CDATA #REQUIRED >
    <!ELEMENT stocks >
    <!ATTLIST stocks
        book IDREF #REQUIRED
        number CDATA #REQUIRED >
    <!ELEMENT written-by >
    <!ATTLIST written-by
        authors IDREFS #REQUIRED >
    <!ELEMENT published-by >
    <!ATTLIST published-by
        publisher IDREF #REQUIRED >
    <!ELEMENT name (#PCDATA )>
    <!ELEMENT address (#PCDATA )>
    <!ELEMENT phone (#PCDATA )>
    <!ELEMENT year (#PCDATA )>
    <!ELEMENT title (#PCDATA )>
    <!ELEMENT price (#PCDATA )>
    <!ELEMENT number (#PCDATA )>
    <!ELEMENT URL (#PCDATA )>
] >
```

Figure 10.1 XML DTD for Bookstore

```
<!DOCTYPE bibliography [
    <!ELEMENT book (title, author+, year, publisher, place?)>
    <!ELEMENT article (title, author+, journal, year, number, volume, pages?)>
    <!ELEMENT author ( last-name, first-name) >
    <!ELEMENT title ( #PCDATA )>
    · · · similar PCDATA declarations for year, publisher, place, journal, year,
        number, volume, pages, last-name and first-name
] >
```

Figure 10.13. DTD for bibliographical data.

```
<bank-2>
    for $c in /bank/customer
    return
        <customer>
            <customer-name> $c/* </customer-name>
            for $a in $c/id(@accounts)
            return $a
        </customer>
</bank-2>
```

10.11 Give a relational schema to represent bibliographical information specified as per the DTD fragment in Figure 10.13. The relational schema must keep track of the order of author elements. You can assume that only books and articles appear as top level elements in XML documents.
Answer:

book (bid, title, year, publisher, place)
article (artid, title, journal, year, number, volume, pages)
book-author (bid, first-name,last-name, order)
article-author (artid, first-name,last-name, order)

10.12 Consider Exercise 10.11, and suppose that authors could also appear as top level elements. What change would have to be done to the relational schema.
Answer:

book (bid, title, year, publisher, place)
article (artid, title, journal, year, number, volume, pages)
author (first-name, last-name)
book-author (bid, first-name, last-name, order)
article-author (artid, first-name,last-name, order)

10.13 Write queries in XQuery on the bibliography DTD fragment in Figure 10.13, to do the following.

- **a.** Find all authors who have authored a book and an article in the same year.
- **b.** Display books and articles sorted by year.
- **c.** Display books with more than one author.

Answer:

a. Find all authors who have authored a book and an article in the same year.

```
for $a in distinct (/bib/book/author),
      $y in /bib/book[author=$a]/year,
      $art in /bib/article[author=$a and year=$y]
return $a
```

b. Display books and articles sorted by year.

```
for $a in ((/bib/book) | (/bib/article))
return $a sortby(year)
```

c. Display books with more than one author.

```
for $b in ((/bib/book[author/count()>1])
return $b
```

10.14 Show the tree representation of the XML data in Figure 10.1, and the representation of the tree using *nodes* and *child* relations described in Section 10.6.1.
Answer: The answer is given in Figure 10.2.

10.15 Consider the following recursive DTD.

```
<!DOCTYPE parts [
      <!ELEMENT part (name, subpartinfo*)>
      <!ELEMENT subpartinfo (part, quantity)>
      <!ELEMENT name ( #PCDATA )>
      <!ELEMENT quantity ( #PCDATA )>
] >
```

a. Give a small example of data corresponding to the above DTD.
b. Show how to map this DTD to a relational schema. You can assume that part names are unique, that is, whereever a part appears, its subpart structure will be the same.

Answer:

a. Give a small example of data corresponding to the above DTD.
The answer is shown in Figure 10.3.
b. Show how to map this DTD to a relational schema.

```
part(partid,name)
subpartinfo(partid, subpartid, qty)
```

Attributes partid and subpartid of subpartinfo are foreign keys to part.

nodes(1,element,bank,–)
nodes(2,element,account,–)
nodes(3,element,account,–)
nodes(4,element,account,–)
nodes(5,element,customer,–)
nodes(6,element,customer,–)
nodes(7,element,depositor,–)
nodes(8,element,depositor,–)
nodes(9,element,depositor,–)
child(2,1) child(3,1) child(4,1)
child(5,1) child(6,1)
child(7,1) child(8,1) child(9,1)
nodes(10,element,account-number,A-101)
nodes(11,element,branch-name,Downtown)
nodes(12,element,balance,500)
child(10,2) child(11,2) child(12,2)
nodes(13,element,account-number,A-102)
nodes(14,element,branch-name,Perryridge)
nodes(15,element,balance,400)
child(13,3) child(14,3) child(15,3)
nodes(16,element,account-number,A-201)
nodes(17,element,branch-name,Brighton)
nodes(18,element,balance,900)
child(16,4) child(17,4) child(18,4)
nodes(19,element,customer-name,Johnson)
nodes(20,element,customer-street,Alma)
nodes(21,element,customer-city,Palo Alto)
child(19,5) child(20,5) child(21,5)
nodes(22,element,customer-name,Hayes)
nodes(23,element,customer-street,Main)
nodes(24,element,customer-city,Harrison)
child(22,6) child(23,6) child(24,6)
nodes(25,element,account-number,A-101)
nodes(26,element,customer-name,Johnson)
child(25,7) child(26,7)
nodes(27,element,account-number,A-201)
nodes(28,element,customer-name,Johnson)
child(27,8) child(28,8)
nodes(29,element,account-number,A-102)
nodes(30,element,customer-name,Hayes)
child(29,9) child(30,9)

Figure 10.2 Relational Representation of XML Data as Trees

```
<parts>
    <part>
        <name> bicycle </name>
        <subpartinfo>
            <part>
                <name> wheel </name>
                <subpartinfo>
                    <part>
                        <name> rim </name>
                    </part>
                    <qty> 1 </qty>
                </subpartinfo>
                <subpartinfo>
                    <part>
                        <name> spokes </name>
                    </part>
                    <qty> 40 </qty>
                </subpartinfo>
                <subpartinfo>
                    <part>
                        <name> tire </name>
                    </part>
                    <qty> 1 </qty>
                </subpartinfo>
            </part>
            <qty> 2 </qty>
        </subpartinfo>
        <subpartinfo>
            <part>
                <name> brake </name>
            </part>
            <qty> 2 </qty>
        </subpartinfo>
        <subpartinfo>
            <part>
                <name> gear </name>
            </part>
            <qty> 3 </qty>
        </subpartinfo>
        <subpartinfo>
            <part>
                <name> frame </name>
            </part>
            <qty> 1 </qty>
        </subpartinfo>
    </part>
</parts>
```

Figure 10.3 Example Parts Data in XML

CHAPTER 11

Storage and File Structure

This chapter presents basic file structure concepts. The chapter really consists of two parts — the first dealing with relational databases, and the second with object-oriented databases. The second part can be omitted without loss of continuity for later chapters.

Many computer science undergraduates have covered some of the material in this chapter in a prior course on data structures or on file structures. Even if students' backgrounds are primarily in data structures, this chapter is still important since it addresses data structure issues as they pertain to disk storage. Buffer management issues, covered in Section 11.5.1 should be familiar to students who have taken an operating systems course. However, there are database-specific aspects of buffer management that make this section worthwhile even for students with an operating system background.

Changes from 3rd edition:

The discussion of storage media, in particular magnetic disks (Section 11.2), has been updated to reflect current technology. The section on RAID structures (Section 11.3) has been improved with examples; the comparison of RAID levels has changed, since disk drive capacity improvements have whittled away at the advantages of RAID 5. Coverage of data dictionaries has been expanded.

Exercises

11.1 List the physical storage media available on the computers you use routinely. Give the speed with which data can be accessed on each medium.
Answer: Your answer will be based on the computers and storage media that you use. Typical examples would be hard disk, floppy disks and CD-ROM drives.

11.2 How does the remapping of bad sectors by disk controllers affect data-retrieval rates?

Answer: Remapping of bad sectors by disk controllers does reduce data retrieval rates because of the loss of sequentiality amongst the sectors. But that is better than the loss of data in case of no remapping!

11.3 Consider the following data and parity-block arrangement on four disks:

Disk 1	Disk 2	Disk 3	Disk 4
B_1	B_2	B_3	B_4
P_1	B_5	B_6	B_7
B_8	P_2	B_9	B_{10}
⋮	⋮	⋮	⋮

The B_i's represent data blocks; the P_i's represent parity blocks. Parity block P_i is the parity block for data blocks B_{4i-3} to B_{4i}. What, if any, problem might this arrangement present?

Answer: This arrangement has the problem that P_i and B_{4i-3} are on the same disk. So if that disk fails, reconstruction of B_{4i-3} is not possible, since data and parity are both lost.

11.4 A power failure that occurs while a disk block is being written could result in the block being only partially written. Assume that partially written blocks can be detected. An atomic block write is one where either the disk block is fully written or nothing is written (i.e., there are no partial writes). Suggest schemes for getting the effect of atomic block writes with the following RAID schemes. Your schemes should involve work on recovery from failure.

a. RAID level 1 (mirroring)
b. RAID level 5 (block interleaved, distributed parity)

Answer:

a. To ensure atomicity, a block write operation is carried out as follows:-
   i. Write the information onto the first physical block.
   ii. When the first write completes successfully, write the same information onto the second physical block.
   iii. The output is declared completed only after the second write completes successfully.

During recovery, each pair of physical blocks is examined. If both are identical and there is no detectable partial-write, then no further actions are necessary. If one block has been partially rewritten, then we replace its contents with the contents of the other block. If there has been no partial-write, but they differ in content, then we replace the contents of the first block with the contents of the second, or vice versa. This recovery procedure ensures that a write to stable storage either succeeds completely (that is, updates both copies) or results in no change.

The requirement of comparing every corresponding pair of blocks during recovery is expensive to meet. We can reduce the cost greatly by keeping track of block writes that are in progress, using a small amount of nonvolatile RAM. On recovery, only blocks for which writes were in progress need to be compared.

b. The idea is similar here. For any block write, the information block is written first followed by the corresponding parity block. At the time of recovery, each set consisting of the n^{th} block of each of the disks is considered. If none of the blocks in the set have been partially-written, and the parity block contents are consistent with the contents of the information blocks, then no further action need be taken. If any block has been partially-written, it's contents are reconstructed using the other blocks. If no block has been partially-written, but the parity block contents do not agree with the information block contents, the parity block's contents are reconstructed.

11.5 RAID systems typically allow you to replace failed disks without stopping access to the system. Thus, the data in the failed disk must be rebuilt and written to the replacement disk while the system is in operation. With which of the RAID levels is the amount of interference between the rebuild and ongoing disk accesses least? Explain your answer.

Answer: RAID level 1 (mirroring) is the one which facilitates rebuilding of a failed disk with minimum interference with the on-going disk accesses. This is because rebuilding in this case involves copying data from just the failed disk's mirror. In the other RAID levels, rebuilding involves reading the entire contents of all the other disks.

11.6 Give an example of a relational-algebra expression and a query-processing strategy in each of the following situations:

a. MRU is preferable to LRU.
b. LRU is preferable to MRU.

Answer:

a. MRU is preferable to LRU where $R_1 \bowtie R_2$ is computed by using a nested-loop processing strategy where each tuple in R_2 must be compared to each block in R_1. After the first tuple of R_2 is processed, the next needed block is the first one in R_1. However, since it is the least recently used, the LRU buffer management strategy would replace that block if a new block was needed by the system.

b. LRU is preferable to MRU where $R_1 \bowtie R_2$ is computed by sorting the relations by join values and then comparing the values by proceeding through the relations. Due to duplicate join values, it may be necessary to "back-up" in one of the relations. This "backing-up" could cross a block boundary into the most recently used block, which would have been replaced by a system using MRU buffer management, if a new block was needed.

Under MRU, some unused blocks may remain in memory forever. In practice, MRU can be used only in special situations like that of the nested-loop strategy discussed in example 0.a

11.7 Consider the deletion of record 5 from the file of Figure 11.8. Compare the relative merits of the following techniques for implementing the deletion:

a. Move record 6 to the space occupied by record 5, and move record 7 to the space occupied by record 6.
b. Move record 7 to the space occupied by record 5.
c. Mark record 5 as deleted, and move no records.

Answer:

a. Although moving record 6 to the space for 5, and moving record 7 to the space for 6, is the most straightforward approach, it requires moving the most records, and involves the most accesses.
b. Moving record 7 to the space for 5 moves fewer records, but destroys any ordering in the file.
c. Marking the space for 5 as deleted preserves ordering and moves no records, but requires additional overhead to keep track of all of the free space in the file. This method may lead to too many "holes" in the file, which if not compacted from time to time, will affect performance because of reduced availability of contiguous free records.

11.8 Show the structure of the file of Figure 11.9 after each of the following steps:

a. Insert (Brighton, A-323, 1600).
b. Delete record 2.
c. Insert (Brighton, A-626, 2000).

Answer: (We use "$\uparrow i$" to denote a pointer to record "i".)

The original file of Figure 11.9.

header	$\uparrow 1$			
record 0		Perryridge	A-102	400
record 1	$\uparrow 4$			
record 2		Mianus	A-215	700
record 3		Downtown	A-101	500
record 4	$\uparrow 6$			
record 5		Perryridge	A-201	900
record 6				
record 7		Downtown	A-110	600
record 8		Perryridge	A-218	700

a. The file after **insert** (Brighton, A-323, 1600).

header	↑ 4			
record 0		Perryridge	A-102	400
record 1		Brighton	A-323	1600
record 2		Mianus	A-215	700
record 3		Downtown	A-101	500
record 4	↑ 6			
record 5		Perryridge	A-201	900
record 6				
record 7		Downtown	A-110	600
record 8		Perryridge	A-218	700

b. The file after **delete** record 2.

header	↑ 2			
record 0		Perryridge	A-102	400
record 1		Brighton	A-323	1600
record 2	↑ 4			
record 3		Downtown	A-101	500
record 4	↑ 6			
record 5		Perryridge	A-201	900
record 6				
record 7		Downtown	A-110	600
record 8		Perryridge	A-218	700

The free record chain could have alternatively been from the header to 4, from 4 to 2, and finally from 2 to 6.

c. The file after **insert** (Brighton, A-626, 2000).

header	↑ 4			
record 0		Perryridge	A-102	400
record 1		Brighton	A-323	1600
record 2		Brighton	A-626	2000
record 3		Downtown	A-101	500
record 4	↑ 6			
record 5		Perryridge	A-201	900
record 6				
record 7		Downtown	A-110	600
record 8		Perryridge	A-218	700

11.9 Give an example of a database application in which the reserved-space method of representing variable-length records is preferable to the pointer method. Explain your answer.

Answer: In the reserved space method, a query comparing the last existing field in a record to some value requires only one read from the disk. This single read is preferable to the potentially many reads needed to chase down the pointers to the last field if the pointer method is used.

11.10 Give an example of a database application in which the pointer method of representing variable-length records is preferable to the reserved-space method. Explain your answer.
Answer: Using the pointer method, a join operation on attributes which are only in the anchor block can be performed on only this smaller amount of data, rather than on the entire relation, as would be the case using the reserved space method. Therefore, in this join example, the pointer method is preferable.

11.11 Show the structure of the file of Figure 11.12 after each of the following steps:

a. Insert (Mianus, A-101, 2800).

b. Insert (Brighton, A-323, 1600).

c. Delete (Perryridge, A-102, 400).

Answer:

a. **insert** (Mianus, A-101, 2800) changes record 2 to:

2	Mianus	A-215	700	A-101	2800	⊥	⊥

b. **insert** (Brighton, A-323, 1600) changes record 5 to:

5	Brighton	A-216	750	A-323	1600	⊥	⊥

c. **delete** (Perryridge, A-102, 400) changes record 0 to:

0	Perryridge	A-102	900	A-218	700	⊥	⊥

11.12 What happens if you attempt to insert the record

(Perryridge, A-929, 3000)

into the file of Figure 11.12?
Answer: Inserting (Perryridge, A-929, 3000) into the file of Figure 11.12 causes an error because the Perryridge record has exceeded the maximum length reserved.

11.13 Show the structure of the file of Figure 11.13 after each of the following steps:

a. Insert (Mianus, A-101, 2800).

b. Insert (Brighton, A-323, 1600).

c. Delete (Perryridge, A-102, 400).

Answer:

a. The figure after **insert** (Mianus, A-101, 2800).

0	↑ 5	Perryridge	A-102	400
1		Round Hill	A-305	350
2	↑ 9	Mianus	A-215	700
3	↑ 7	Downtown	A-101	500
4		Redwood	A-222	700
5	↑ 8		A-201	900
6		Brighton	A-216	750
7			A-110	600
8			A-218	700
9			A-101	2800

b. The figure after **insert** (Brighton, A-323, 1600).

0	↑ 5	Perryridge	A-102	400
1		Round Hill	A-305	350
2	↑ 9	Mianus	A-215	700
3	↑ 7	Downtown	A-101	500
4		Redwood	A-222	700
5	↑ 8		A-201	900
6	↑ 10	Brighton	A-216	750
7			A-110	600
8			A-218	700
9			A-101	2800
10			A-323	1600

c. The figure after **delete** (Perryridge, A-102, 400).

1		Round Hill	A-305	350
2	↑ 9	Mianus	A-215	700
3	↑ 7	Downtown	A-101	500
4		Redwood	A-222	700
5	↑ 8	Perryridge	A-201	900
6	↑ 10	Brighton	A-216	750
7			A-110	600
8			A-218	700
9			A-101	2800
10			A-323	1600

11.14 Explain why the allocation of records to blocks affects database-system performance significantly.

Answer: If we allocate related records to blocks, we can often retrieve most, or all, of the requested records by a query with one disk access. Disk accesses tend to be the bottlenecks in databases; since this allocation strategy reduces the number of disk accesses for a given operation, it significantly improves performance.

11.15 If possible, determine the buffer-management strategy used by the operating system running on your local computer system, and what mechanisms it provides to control replacement of pages. Discuss how the control on replacement

that it provides would be useful for the implementation of database systems.
Answer: The typical OS uses LRU for buffer replacement. This is often a bad strategy for databases. As explained in Section 11.5.2 of the text, MRU is the best strategy for nested loop join. In general no single strategy handles all scenarios well, and ideally the database system should be given its own buffer cache for which the replacement policy takes into account all the performance related issues.

11.16 In the sequential file organization, why is an overflow *block* used even if there is, at the moment, only one overflow record?
Answer: An overflow block is used in sequential file organization because a block is the smallest space which can be read from a disk. Therefore, using any smaller region would not be useful from a performance standpoint. The space saved by allocating disk storage in record units would be overshadowed by the performance cost of allowing blocks to contain records of multiple files.

11.17 List two advantages and two disadvantages of each of the following strategies for storing a relational database:

a. Store each relation in one file.
b. Store multiple relations (perhaps even the entire database) in one file.

Answer:

a. Advantages of storing a relation as a file include using the file system provided by the OS , thus simplifying the DBMS, but incurs the disadvantage of restricting the ability of the DBMS to increase performance by using more sophisticated storage structures.
b. By using one file for the entire database, these complex structures can be implemented through the DBMS, but this increases the size and complexity of the DBMS.

11.18 Consider a relational database with two relations:

course (*course-name, room, instructor*)
enrollment (*course-name, student-name, grade*)

Define instances of these relations for three courses, each of which enrolls five students. Give a file structure of these relations that uses clustering.

Answer:

course relation

course-name	*room*	*instructor*	
Pascal	CS-101	Calvin, B	c_1
C	CS-102	Calvin, B	c_2
LISP	CS-102	Kess, J	c_3

enrollment relation

course-name	*student-name*	*grade*	
Pascal	Carper, D	A	e_1
Pascal	Merrick, L	A	e_2
Pascal	Mitchell, N	B	e_3
Pascal	Bliss, A	C	e_4
Pascal	Hames, G	C	e_5
C	Nile, M	A	e_6
C	Mitchell, N	B	e_7
C	Carper, D	A	e_8
C	Hurly, I	B	e_9
C	Hames, G	A	e_{10}
Lisp	Bliss, A	C	e_{11}
Lisp	Hurly, I	B	e_{12}
Lisp	Nile, M	D	e_{13}
Lisp	Stars, R	A	e_{14}
Lisp	Carper, D	A	e_{15}

Block 0 contains: c_1, e_1, e_2, e_3, e_4, and e_5
Block 1 contains: c_2, e_6, e_7, e_8, e_9 and e_{10}
Block 2 contains: c_3, e_{11}, e_{12}, e_{13}, e_{14}, and e_{15}

11.19 Consider the following bitmap technique for tracking free space in a file. For each block in the file, two bits are maintained in the bitmap. If the block is between 0 and 30 percent full the bits are 00, between 30 and 60 percent the bits are 01, between 60 and 90 percent the bits are 10, and above 90 percent the bits are 11. Such bitmaps can be kept in memory even for quite large files.

a. Describe how to keep the bitmap up-to-date on record insertions and deletions.

b. Outline the benefit of the bitmap technique over free lists when searching for free space and when updating free space information.

Answer:

a. Everytime a record is inserted/deleted, check if the usage of the block has changed levels. In that case, update the corrosponding bits. Note that we don't need to access the bitmaps at all unless the usage crosses a boundary, so in most of the cases there is no overhead.

b. When free space for a large record or a set of records is sought, then multiple free list entries may have to be scanned before finding a proper sized one, so overheads are much higher. With bitmaps, one page of bitmap can store free info for many pages, so IO spent for finding free space is minimal. Similarly, when a whole block or a large part of it is deleted, bitmap technique is more convenient for updating free space information.

11.20 Give a normalized version of the *Index-metadata* relation, and explain why using the normalized version would result in worse performance.

Answer: The *Index-metadata* relation can be normalized as follows

Index-metadata (index-name, relation-name, index-type, attrib-set)
Attribset-metadata (relation-name, attrib-set, attribute-name)

Though the normalized version will have less space requirements, it will require extra disk accesses to read *Attribset-metadata* everytime an index has to be accessed. Thus, it will lead to worse performance.

11.21 Explain why a physical OID must contain more information than a pointer to a physical storage location.
Answer: A physical OID needs to have a *unique identifier* in addition to a pointer to a physical storage location. This is required to prevent dereferences of dangling pointers.

11.22 If physical OIDs are used, an object can be relocated by keeping a forwarding pointer to its new location. In case an object gets forwarded multiple times, what would be the effect on retrieval speed? Suggest a technique to avoid multiple accesses in such a case.
Answer:

If an object gets forwarded multiple times, the retrieval speed will decrease because accessing it will require accessing the series of locations from which the object has been successively forwarded to the current location.

To avoid multiple accesses, whenever an object is accessed using an old pointer, update each pointer in the path to point to the current location of the object. With this path compression, whenever the object is accessed again through any pointer in that path, the object can be directly reached.

11.23 Define the term *dangling pointer*. Describe how the unique-id scheme helps in detecting dangling pointers in an object-oriented database.
Answer: A *dangling pointer* is a pointer to an area which no longer contains valid data.

In the unique-id scheme to detect dangling pointers, physical OIDs may contain a *unique identifier* which is an integer that distinguishes the OID from the identifiers of other objects that happened to be stored at the same location earlier, and were deleted or moved elsewhere. The unique identifier is also stored with the object, and the identifiers in an OID and the corresponding object should match. If the unique identifier in a physical OID does not match the unique identifier in the object to which that OID points, the system detects that the pointer is a dangling pointer, and signals an error.

11.24 Consider the example on page 435, which shows that there is no need for deswizzling if hardware swizzling is used. Explain why, in that example, it is safe to change the short identifier of page 679.34278 from 2395 to 5001. Can some other page already have short identifier 5001? If it could, how can you handle that situation?
Answer: While swizzling, if the short identifier of page 679.34278 is changed from 2395 to 5001, it is either because

a. the system discovers that 679.34278 has already been allocated the virtual-memory page 5001 in some previous step, or else
b. 679.34278 has not been allocated any virtual memory page so far, and the free virtual memory page 5001 is now allocated to it.

Thus in either case, it cannot be true that the current page already uses the same short identifier 5001 to refer to some database page other than 679.34278. Some other page may use 5001 to refer to a different database page, but then each page has its own independent mapping from short to full page identifiers, so this is all right.

Note that if we do swizzling as described in the text, and different processes need simultaneous access to a database page, they will have to map separate copies of the page to their individual virtual address spaces. Extensions to the scheme are possible to avoid this.

CHAPTER 12

Indexing and Hashing

This chapter covers indexing techniques ranging from the most basic one to highly specialized ones. Due to the extensive use of indices in database systems, this chapter constitutes an important part of a database course.

A class that has already had a course on data-structures would likely be familiar with hashing and perhaps even B^+-trees. However, this chapter is necessary reading even for those students since data structures courses typically cover indexing in main memory. Although the concepts carry over to database access methods, the details (e.g., block-sized nodes), will be new to such students.

The sections on B-trees (Sections 12.4), grid files (Section 12.9.3) and bitmap indexing (Section 12.9.4) may be omitted if desired.

Changes from 3rd edition:
The description of querying on B^+-trees has been augmented with pseudo-code. The pseudo-code for insertion on B^+-trees has been simplified. The section on index definition in SQL (Section 12.8) is new to this edition, as is the coverage of bitmap indices (Section 12.9.4).

Exercises

12.1 When is it preferable to use a dense index rather than a sparse index? Explain your answer.
Answer: It is preferable to use a dense index instead of a sparse index when the file is not sorted on the indexed field (such as when the index is a secondary index) or when the index file is small compared to the size of memory.

12.2 Since indices speed query processing, why might they not be kept on several search keys? List as many reasons as possible.
Answer: Reasons for not keeping several search indices include:

a. Every index requires additional CPU time and disk I/O overhead during inserts and deletions.
b. Indices on non-primary keys might have to be changed on updates, although an index on the primary key might not (this is because updates typically do not modify the primary key attributes).
c. Each extra index requires additional storage space.
d. For queries which involve conditions on several search keys, efficiency might not be bad even if only some of the keys have indices on them. Therefore database performance is improved less by adding indices when many indices already exist.

12.3 What is the difference between a primary index and a secondary index?
Answer: The primary index is on the field which specifies the sequential order of the file. There can be only one primary index while there can be many secondary indices.

12.4 Is it possible in general to have two primary indices on the same relation for different search keys? Explain your answer.
Answer: In general, it is not possible to have two primary indices on the same relation for different keys because the tuples in a relation would have to be stored in different order to have same values stored together. We could accomplish this by storing the relation twice and duplicating all values, but for a centralized system, this is not efficient.

12.5 Construct a B^+-tree for the following set of key values:

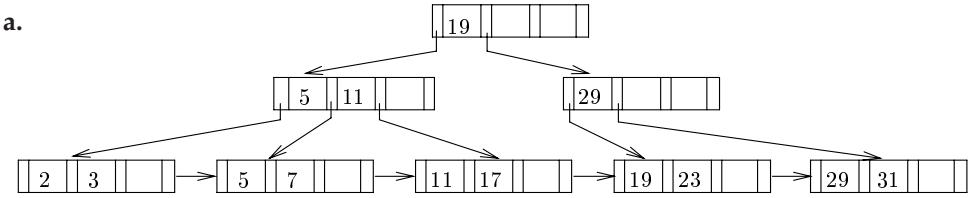
(2, 3, 5, 7, 11, 17, 19, 23, 29, 31)

Assume that the tree is initially empty and values are added in ascending order. Construct B^+-trees for the cases where the number of pointers that will fit in one node is as follows:

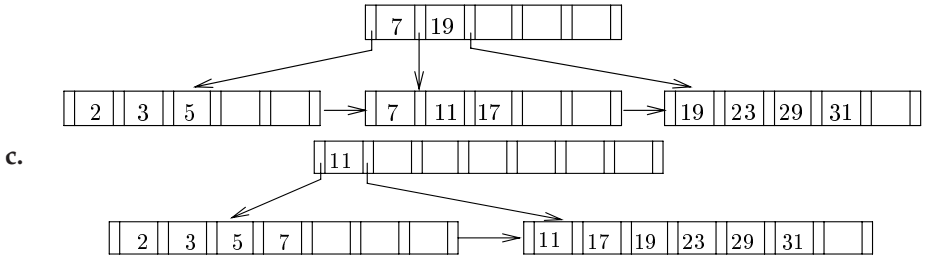
a. Four
b. Six
c. Eight

Answer: The following were generated by inserting values into the B^+-tree in ascending order. A node (other than the root) was never allowed to have fewer than $\lceil n/2 \rceil$ values/pointers.

a.



b.



12.6 For each B^+-tree of Exercise 12.5, show the steps involved in the following queries:

a. Find records with a search-key value of 11.
b. Find records with a search-key value between 7 and 17, inclusive.

Answer:

With structure 0.a:

a. Find records with a value of 11
   i. Search the first level index; follow the first pointer.
   ii. Search next level; follow the third pointer.
   iii. Search leaf node; follow first pointer to records with key value 11.
b. Find records with value between 7 and 17 (inclusive)
   i. Search top index; follow first pointer.
   ii. Search next level; follow second pointer.
   iii. Search third level; follow second pointer to records with key value 7, and after accessing them, return to leaf node.
   iv. Follow fourth pointer to next leaf block in the chain.
   v. Follow first pointer to records with key value 11, then return.
   vi. Follow second pointer to records with with key value 17.

With structure 0.b:

a. Find records with a value of 11
   i. Search top level; follow second pointer.
   ii. Search next level; follow second pointer to records with key value 11.
b. Find records with value between 7 and 17 (inclusive)
   i. Search top level; follow second pointer.
   ii. Search next level; follow first pointer to records with key value 7, then return.
   iii. Follow second pointer to records with key value 11, then return.
   iv. Follow third pointer to records with key value 17.

With structure 0.c:

a. Find records with a value of 11
   i. Search top level; follow second pointer.
   ii. Search next level; follow first pointer to records with key value 11.
b. Find records with value between 7 and 17 (inclusive)

i. Search top level; follow first pointer.
ii. Search next level; follow fourth pointer to records with key value 7, then return.
iii. Follow eighth pointer to next leaf block in chain.
iv. Follow first pointer to records with key value 11, then return.
v. Follow second pointer to records with key value 17.

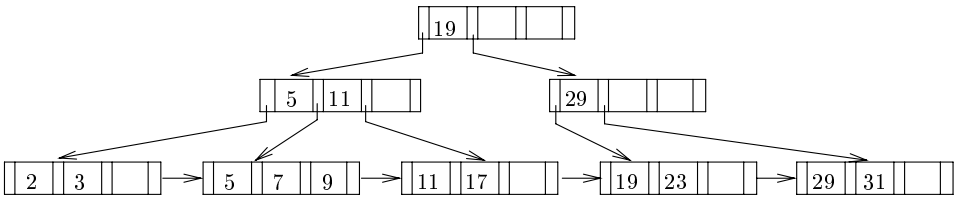
12.7 For each B^+-tree of Exercise 12.5, show the form of the tree after each of the following series of operations:

a. Insert 9.
b. Insert 10.
c. Insert 8.
d. Delete 23.
e. Delete 19.

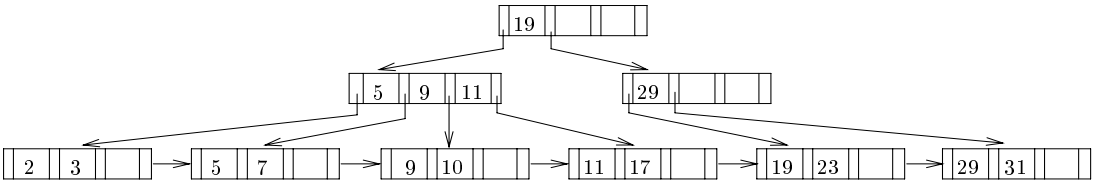
Answer:

- With structure 0.a:

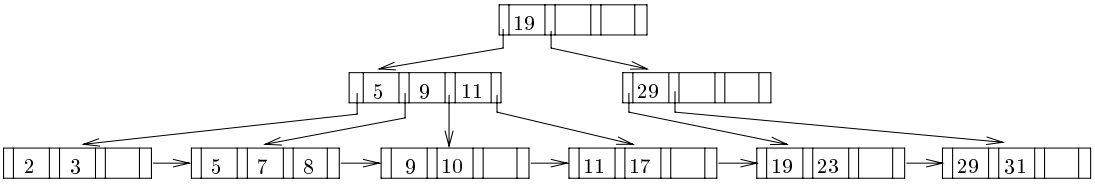
Insert 9:



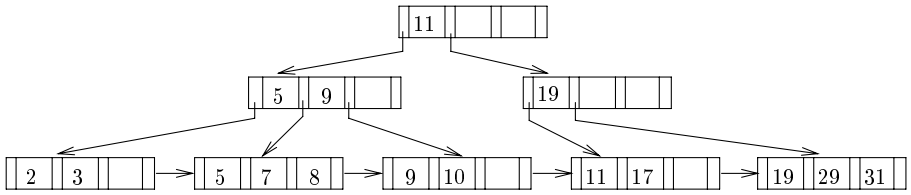
Insert 10:



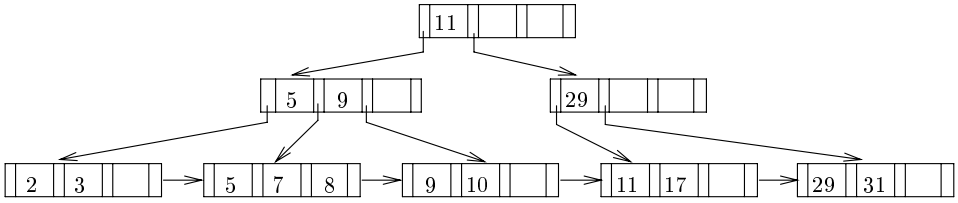
Insert 8:



Delete 23:

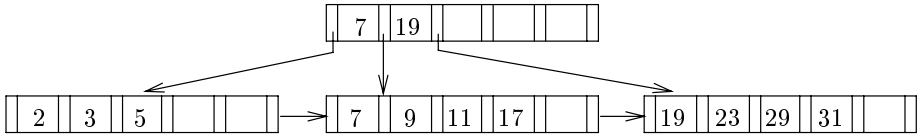


Delete 19:

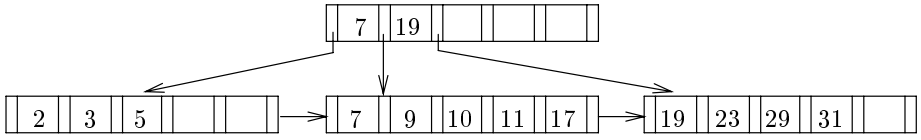


- With structure 0.b:

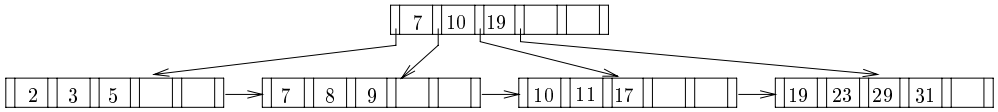
  Insert 9:



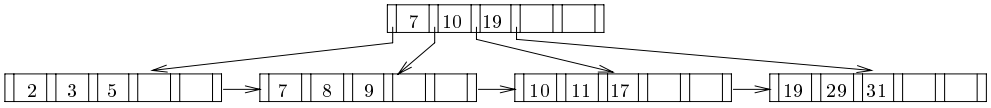
  Insert 10:



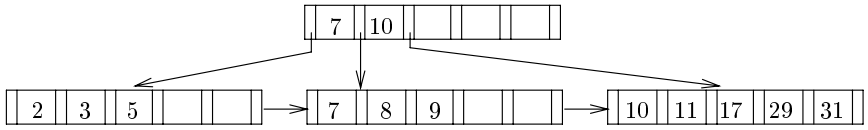
  Insert 8:



  Delete 23:

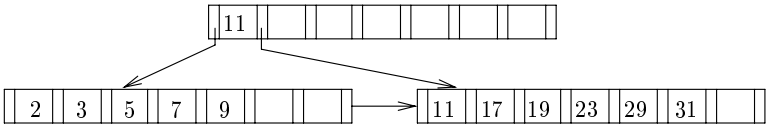


  Delete 19:

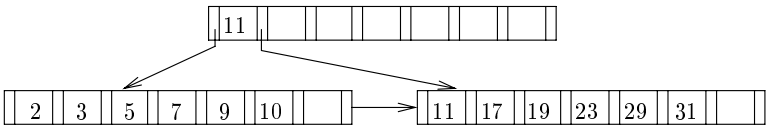


- With structure 0.c:

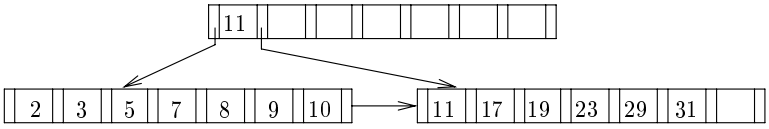
Insert 9:



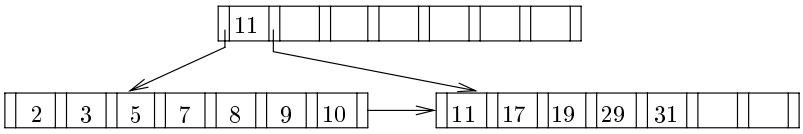
Insert 10:



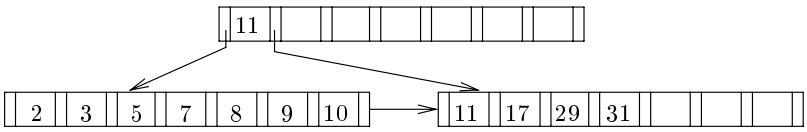
Insert 8:



Delete 23:



Delete 19:



12.8 Consider the modified redistribution scheme for B$^+$-trees described in page 463. What is the expected height of the tree as a function of n?
Answer: If there are K search-key values and $m - 1$ siblings are involved in the redistribution, the expected height of the tree is: $log_{\lfloor (m-1)n/m \rfloor}(K)$

12.9 Repeat Exercise 12.5 for a B-tree.
Answer: The algorithm for insertion into a B-tree is:

Locate the leaf node into which the new key-pointer pair should be inserted. If there is space remaining in that leaf node, perform the insertion at the correct location, and the task is over. Otherwise insert the key-pointer pair conceptually into the correct location in the leaf node, and then split it along the middle. The middle key-pointer pair does not go into either of the resultant nodes of the split operation. Instead it is inserted into the parent node, along with the tree pointer to the new child. If there is no space in the parent, a similar procedure is repeated.

The deletion algorithm is:

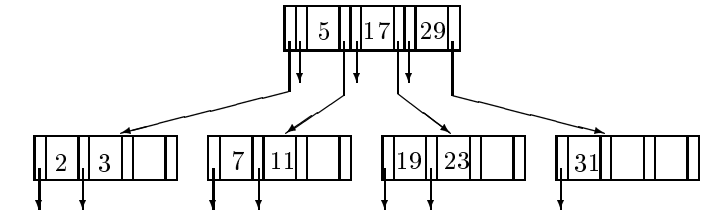
Locate the key value to be deleted, in the B-tree.

a. If it is found in a leaf node, delete the key-pointer pair, and the record from the file. If the leaf node contains less than $\lceil n/2 \rceil - 1$ entries as a result of this deletion, it is either merged with its siblings, or some entries are redistributed to it. Merging would imply a deletion, whereas redistribution would imply change(s) in the parent node's entries. The deletions may ripple upto the root of the B-tree.

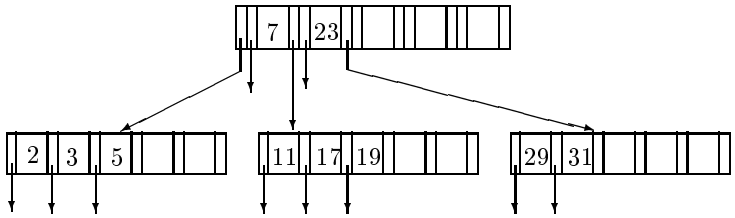
b. If the key value is found in an internal node of the B-tree, replace it and its record pointer by the smallest key value in the subtree immediately to its right and the corresponding record pointer. Delete the actual record in the database file. Then delete that smallest key value-pointer pair from the subtree. This deletion may cause further rippling deletions till the root of the B-tree.

Below are the B-trees we will get after insertion of the given key values. We assume that leaf and non-leaf nodes hold the same number of search key values.

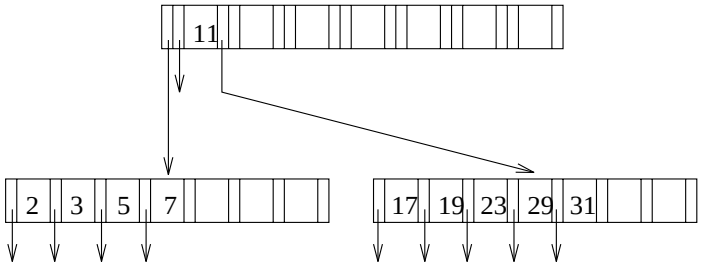
a.



b.



c.



12.10 Explain the distinction between closed and open hashing. Discuss the relative merits of each technique in database applications.

Answer: Open hashing may place keys with the same hash function value in different buckets. Closed hashing always places such keys together in the same bucket. Thus in this case, different buckets can be of different sizes, though the

implementation may be by linking together fixed size buckets using overflow chains. Deletion is difficult with open hashing as *all* the buckets may have to inspected before we can ascertain that a key value has been deleted, whereas in closed hashing only that bucket whose address is obtained by hashing the key value need be inspected. Deletions are more common in databases and hence closed hashing is more appropriate for them. For a small, static set of data lookups may be more efficient using open hashing. The symbol table of a compiler would be a good example.

12.11 What are the causes of bucket overflow in a hash file organization? What can be done to reduce the occurrence of bucket overflows?
Answer: The causes of bucket overflow are :-

- **a.** Our estimate of the number of records that the relation will have was too low, and hence the number of buckets allotted was not sufficient.
- **b.** Skew in the distribution of records to buckets. This may happen either because there are many records with the same search key value, or because the the hash function chosen did not have the desirable properties of uniformity and randomness.

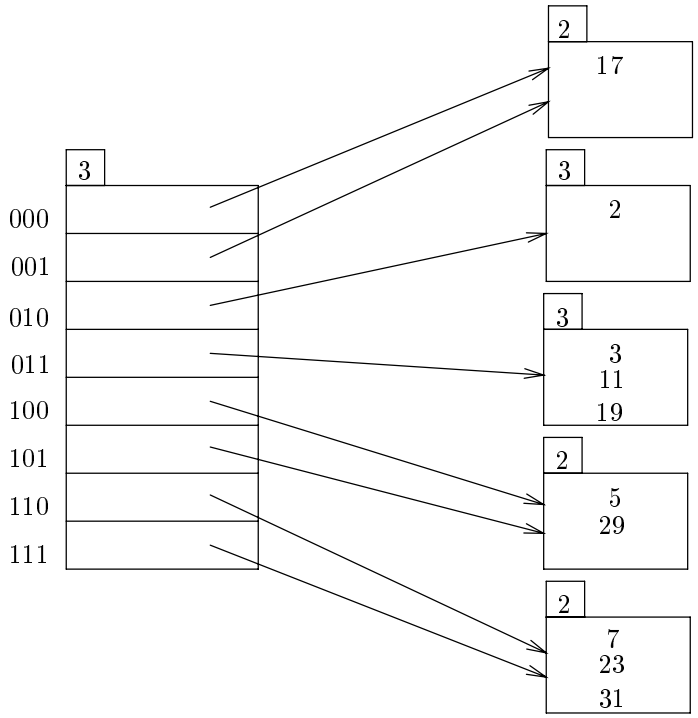
To reduce the occurrence of overflows, we can :-

- **a.** Choose the hash function more carefully, and make better estimates of the relation size.
- **b.** If the estimated size of the relation is n_r and number of records per block is f_r, allocate $(n_r/f_r) * (1 + d)$ buckets instead of (n_r/f_r) buckets. Here d is a fudge factor, typically around 0.2. Some space is wasted: About 20 percent of the space in the buckets will be empty. But the benefit is that some of the skew is handled and the probability of overflow is reduced.

12.12 Suppose that we are using extendable hashing on a file that contains records with the following search-key values:

2, 3, 5, 7, 11, 17, 19, 23, 29, 31

Show the extendable hash structure for this file if the hash function is $h(x) = x$ mod 8 and buckets can hold three records.
Answer:

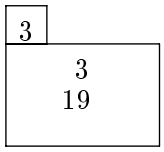


12.13 Show how the extendable hash structure of Exercise 12.12 changes as the result of each of the following steps:

a. Delete 11.
b. Delete 31.
c. Insert 1.
d. Insert 15.

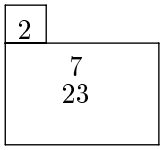
Answer:

a. Delete 11: From the answer to Exercise 12.12, change the third bucket to:

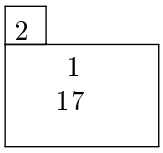


At this stage, it is possible to coalesce the second and third buckets. Then it is enough if the bucket address table has just four entries instead of eight. For the purpose of this answer, we do not do the coalescing.

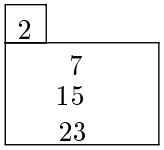
b. Delete 31: From the answer to 12.12, change the last bucket to:



c. Insert 1: From the answer to 12.12, change the first bucket to:



d. Insert 15: From the answer to 12.12, change the last bucket to:



12.14 Give pseudocode for deletion of entries from an extendable hash structure, including details of when and how to coalesce buckets. Do not bother about reducing the size of the bucket address table.
Answer: Let i denote the number of bits of the hash value used in the hash table. Let **BSIZE** denote the maximum capacity of each bucket.

```
delete(value K_l)
begin
    j = first i high-order bits of h(K_l);
    delete value K_l from bucket j;
    coalesce(bucket j);
end

coalesce(bucket j)
begin
    i_j = bits used in bucket j;
    k = any bucket with first (i_j − 1) bits same as that
        of bucket j while the bit i_j is reversed;
    i_k = bits used in bucket k;
    if(i_j ≠ i_k)
        return; /* buckets cannot be merged */
    if(entries in j + entries in k > BSIZE)
        return; /* buckets cannot be merged */
    move entries of bucket k into bucket j;

    decrease the value of i_j by 1;
    make all the bucket-address-table entries,
    which pointed to bucket k, point to j;

    coalesce(bucket j);
end
```

Note that we can only merge two buckets at a time. The common hash prefix of the resultant bucket will have length one less than the two buckets merged. Hence we look at the buddy bucket of bucket j differing from it only at the last bit. If the common hash prefix of this bucket is not i_j, then this implies that the buddy bucket has been further split and merge is not possible.

When merge is successful, further merging may be possible, which is handled by a recursive call to *coalesce* at the end of the function.

12.15 Suggest an efficient way to test if the bucket address table in extendable hashing can be reduced in size, by storing an extra count with the bucket address table. Give details of how the count should be maintained when buckets are split, coalesced or deleted.

(Note: Reducing the size of the bucket address table is an expensive operation, and subsequent inserts may cause the table to grow again. Therefore, it is best not to reduce the size as soon as it is possible to do so, but instead do it only if the number of index entries becomes small compared to the bucket address table size.)

Answer: If the hash table is currently using i bits of the hash value, then maintain a count of buckets for which the length of common hash prefix is exactly i.

Consider a bucket j with length of common hash prefix i_j. If the bucket is being split, and i_j is equal to i, then reset the count to 1. If the bucket is being split and i_j is one less that i, then increase the count by 1. It the bucket if being coalesced, and i_j is equal to i then decrease the count by 1. If the count becomes 0, then the bucket address table can be reduced in size at that point.

However, note that if the bucket address table is not reduced at that point, then the count has no significance afterwards. If we want to postpone the reduction, we have to keep an array of counts, i.e. a count for each value of common hash prefix. The array has to be updated in a similar fashion. The bucket address table can be reduced if the i^{th} entry of the array is 0, where i is the number of bits the table is using. Since bucket table reduction is an expensive operation, it is not always advisable to reduce the table. It should be reduced only when sufficient number of entries at the end of count array become 0.

12.16 Why is a hash structure not the best choice for a search key on which range queries are likely?

Answer: A range query cannot be answered efficiently using a hash index, we will have to read all the buckets. This is because key values in the range do not occupy consecutive locations in the buckets, they are distributed uniformly and randomly throughout all the buckets.

12.17 Consider a grid file in which we wish to avoid overflow buckets for performance reasons. In cases where an overflow bucket would be needed, we instead reorganize the grid file. Present an algorithm for such a reorganization.

Answer: Let us consider a two-dimensional grid array. When a bucket overflows, we can split the ranges corresponding to that row and column into two, in both the linear scales. Thus the linear scales will get one additional entry each, and the bucket is split into four buckets. The ranges should be split in such a way as to ensure that the four resultant buckets have nearly the same number of values.

There can be several other heuristics for deciding how to reorganize the ranges, and hence the linear scales and grid array.

12.18 Consider the *account* relation shown in Figure 12.25.

- **a.** Construct a bitmap index on the attributes *branch-name* and *balance*, dividing *balance* values into 4 ranges: below 250, 250 to below 500, 500 to below 750, and 750 and above.
- **b.** Consider a query that requests all accounts in Downtown with a balance of 500 or more. Outline the steps in answering the query, and show the final and intermediate bitmaps constructed to answer the query.

Answer: We reproduce the account relation of Figure 12.25 below.

A-217	Brighton	750
A-101	Downtown	500
A-1 10	Downtown	600
A-215	Mianus	700
A-102	Perryridge	400
A-201	Perryridge	900
A-218	Perryridge	700
A-222	Redwood	700
A-305	Round Hill	350

Bitmaps for *branch-name*

Brighton	1	0	0	0	0	0	0	0	0
Downtown	0	1	1	0	0	0	0	0	0
Mianus	0	0	0	1	0	0	0	0	0
Perryridge	0	0	0	0	1	1	1	0	0
Redwood	0	0	0	0	0	0	0	1	0
Round hill	0	0	0	0	0	0	0	0	1

Bitmaps for *balance*

L_1	0	0	0	0	0	0	0	0	0
L_2	0	0	0	0	1	0	0	0	1
L_3	0	1	1	1	0	0	1	1	0
L_4	1	0	0	0	0	1	0	0	0

where, level L_1 is below 250, level L_2 is from 250 to below 500, L_3 from 500 to below 750 and level L_4 is above 750.

To find all accounts in Downtown with a balance of 500 or more, we find the union of bitmaps for levels L_3 and L_4 and then intersect it with the bitmap for Downtown.

Downtown	0	1	1	0	0	0	0	0	0
L_3	0	1	1	1	0	0	1	1	0
L_4	1	0	0	0	0	1	0	0	0
$L_3 \cup L_4$	1	1	1	1	0	1	1	1	0
Downtown	0	1	1	0	0	0	0	0	0
Downtown $\cap (L_3 \cup L_4)$	0	1	1	0	0	0	0	0	0

Thus, the required tuples are A-101 and A-110.

12.19 Show how to compute existence bitmaps from other bitmaps. Make sure that your technique works even in the presence of null values, by using a bitmap for the value *null*.

Answer: The existence bitmap for a relation can be calculated by taking the

union (logical-or) of all the bitmaps on that attribute, including the bitmap for value *null*.

12.20 How does data encryption affect index schemes? In particular, how might it affect schemes that attempt to store data in sorted order?

Answer: Note that indices must operate on the encrypted data or someone could gain access to the index to interpret the data. Otherwise, the index would have to be restricted so that only certain users could access it. To keep the data in sorted order, the index scheme would have to decrypt the data at each level in a tree. Note that hash systems would not be affected.

CHAPTER 13

Query Processing

This chapter describes the process by which queries are executed efficiently by a database system. The chapter starts off with measures of cost, then proceeds to algorithms for evaluation of relational algebra operators and expressions. This chapter applies concepts from Chapters 3, 11, and 12.

Changes from 3rd edition:

The single chapter on query processing in the previous edition has been replaced by two chapters, the first on query processing and the second on query optimization. Another significant change is the separation of size estimation from the presentation of query processing algorithms.

As a result, of these changes, query processing algorithms can be covered without tedious and distracting details of size estimation. Although size estimation is covered later, in Chapter 14, the presentation there has been simplified by omitting some details. Instructors can choose to cover query processing but omit query optimization, without loss of continuity with later chapters.

Exercises

13.1 Why is it not desirable to force users to make an explicit choice of a query-processing strategy? Are there cases in which it *is* desirable for users to be aware of the costs of competing query-processing strategies? Explain your answer.

Answer: In general it is not desirable to force users to choose a query processing strategy because naive users might select an inefficient strategy. The reason users would make poor choices about processing queries is that they would not know how a relation is stored, nor about its indices. It is unreasonable to force users to be aware of these details since ease of use is a major object

of database query languages. If users are aware of the costs of different strategies they could write queries efficiently, thus helping performance. This could happen if experts were using the system.

13.2 Consider the following SQL query for our bank database:

select *T.branch-name*
from *branch T, branch S*
where *T.assets* > *S.assets* **and** *S.branch-city* = "Brooklyn"

Write an efficient relational-algebra expression that is equivalent to this query. Justify your choice.
Answer:

$$\Pi_{T.branch\text{-}name}((\Pi_{branch\text{-}name,\, assets}(\rho_T(branch))) \bowtie_{T.assets > S.assets} (\Pi_{assets}\ (\sigma_{(branch\text{-}city = \text{“}Brooklyn\text{”})}(\rho_S(branch)))))$$

This expression performs the theta join on the smallest amount of data possible. It does this by restricting the right hand side operand of the join to only those branches in Brooklyn, and also eliminating the unneeded attributes from both the operands.

13.3 What are the advantages and disadvantages of hash indices relative to B^+-tree indices? How might the type of index available influence the choice of a query-processing strategy?
Answer: Hash indices enable us to perform point lookup (eg. $\sigma_{A=r}(relation)$) operations very fast, but for range searches the B^+-tree index would be much more efficient. If there is a range query to be evaluated, and only a hash index is available, the better strategy might be to perform a file scan rather than using that index.

13.4 Assume (for simplicity in this exercise) that only one tuple fits in a block and memory holds at most 3 page frames. Show the runs created on each pass of the sort-merge algorithm, when applied to sort the following tuples on the first attribute: (kangaroo, 17), (wallaby, 21), (emu, 1), (wombat, 13), (platypus, 3), (lion, 8), (warthog, 4), (zebra, 11), (meerkat, 6), (hyena, 9), (hornbill, 2), (baboon, 12).
Answer: We will refer to the tuples (kangaroo, 17) through (baboon, 12) using tuple numbers t_1 through t_{12}. We refer to the j^{th} run used by the i^{th} pass, as r_{ij}. The initial sorted runs have three blocks each. They are:-

$$\begin{aligned} r_{11} &= \{t_3, t_1, t_2\} \\ r_{12} &= \{t_6, t_5, t_4\} \\ r_{13} &= \{t_9, t_7, t_8\} \\ r_{14} &= \{t_{12}, t_{11}, t_{10}\} \end{aligned}$$

Each pass merges three runs. Therefore the runs after the end of the first pass are:-

$$\begin{aligned} r_{21} &= \{t_3, t_1, t_6, t_9, t_5, t_2, t_7, t_4, t_8\} \\ r_{22} &= \{t_{12}, t_{11}, t_{10}\} \end{aligned}$$

At the end of the second pass, the tuples are completely sorted into one run:-

$$r_{31} = \{t_{12}, t_3, t_{11}, t_{10}, t_1, t_6, t_9, t_5, t_2, t_7, t_4, t_8\}$$

13.5 Let relations $r_1(A, B, C)$ and $r_2(C, D, E)$ have the following properties: r_1 has 20,000 tuples, r_2 has 45,000 tuples, 25 tuples of r_1 fit on one block, and 30 tuples of r_2 fit on one block. Estimate the number of block accesses required, using each of the following join strategies for $r_1 \bowtie r_2$:

a. Nested-loop join
b. Block nested-loop join
c. Merge join
d. Hash join

Answer: r_1 needs 800 blocks, and r_2 needs 1500 blocks. Let us assume M pages of memory. If $M > 800$, the join can easily be done in 1500 + 800 disk accesses, using even plain nested-loop join. So we consider only the case where $M \leq 800$ pages.

a. Nested-loop join:
Using r_1 as the outer relation we need $20000 * 1500 + 800 = 30,000,800$ disk accesses, if r_2 is the outer relation we need $45000 * 800 + 1500 = 36,001,500$ disk accesses.

b. Block nested-loop join:
If r_1 is the outer relation, we need $\lceil \frac{800}{M-1} \rceil * 1500 + 800$ disk accesses, if r_2 is the outer relation we need $\lceil \frac{1500}{M-1} \rceil * 800 + 1500$ disk accesses.

c. Merge-join:
Assuming that r_1 and r_2 are not initially sorted on the join key, the total sorting cost inclusive of the output is $B_s = 1500(2\lceil log_{M-1}(1500/M) \rceil + 2) + 800(2\lceil log_{M-1}(800/M) \rceil + 2)$ disk accesses. Assuming all tuples with the same value for the join attributes fit in memory, the total cost is $B_s + 1500 + 800$ disk accesses.

d. Hash-join:
We assume no overflow occurs. Since r_1 is smaller, we use it as the build relation and r_2 as the probe relation. If $M > 800/M$, i.e. no need for recursive partitioning, then the cost is $3(1500 + 800) = 6900$ disk accesses, else the cost is $2(1500 + 800)\lceil log_{M-1}(800) - 1 \rceil + 1500 + 800$ disk accesses.

13.6 Design a variant of the hybrid merge–join algorithm for the case where both relations are not physically sorted, but both have a sorted secondary index on the join attributes.
Answer: We merge the leaf entries of the first sorted secondary index with

the leaf entries of the second sorted secondary index. The result file contains pairs of addresses, the first address in each pair pointing to a tuple in the first relation, and the second address pointing to a tuple in the second relation.

This result file is first sorted on the first relation's addresses. The relation is then scanned in physical storage order, and addresses in the result file are replaced by the actual tuple values. Then the result file is sorted on the second relation's addresses, allowing a scan of the second relation in physical storage order to complete the join.

13.7 The indexed nested-loop join algorithm described in Section 13.5.3 can be inefficient if the index is a secondary index, and there are multiple tuples with the same value for the join attributes. Why is it inefficient? Describe a way, using sorting, to reduce the cost of retrieving tuples of the inner relation. Under what conditions would this algorithm be more efficient than hybrid merge–join?
Answer: If there are multiple tuples in the inner relation with the same value for the join attributes, we may have to access that many blocks of the inner relation for each tuple of the outer relation. That is why it is inefficient. To reduce this cost we can perform a join of the outer relation tuples with just the secondary index leaf entries, postponing the inner relation tuple retrieval. The result file obtained is then sorted on the inner relation addresses, allowing an efficient physical order scan to complete the join.

Hybrid merge–join requires the outer relation to be sorted. The above algorithm does not have this requirement, but for each tuple in the outer relation it needs to perform an index lookup on the inner relation. If the outer relation is much larger than the inner relation, this index lookup cost will be less than the sorting cost, thus this algorithm will be more efficient.

13.8 Estimate the number of block accesses required by your solution to Exercise 13.6 for $r_1 \bowtie r_2$, where r_1 and r_2 are as defined in Exercise 13.5.
Answer: r_1 occupies 800 blocks, and r_2 occupies 1500 blocks. Let there be n pointers per index leaf block (we assume that both the indices have leaf blocks and pointers of equal sizes). Let us assume M pages of memory, $M < 800$. r_1's index will need $B_1 = \lceil \frac{20000}{n} \rceil$ leaf blocks, and r_2's index will need $B_2 = \lceil \frac{45000}{n} \rceil$ leaf blocks. Therefore the merge join will need $B_3 = B_1 + B_2$ accesses, without output. The number of output tuples is estimated as $n_o = \lceil \frac{20000 * 45000}{max(V(C,r_1),V(C,r_2))} \rceil$. Each output tuple will need two pointers, so the number of blocks of join output will be $B_{o1} = \lceil \frac{n_o}{n/2} \rceil$. Hence the join needs $B_j = B_3 + B_{o1}$ disk block accesses.

Now we have to replace the pointers by actual tuples. For the first sorting, $B_{s1} = B_{o1}(2\lceil log_{M-1}(B_{o1}/M) \rceil + 2)$ disk accesses are needed, including the writing of output to disk. The number of blocks of r_1 which have to be accessed in order to replace the pointers with tuple values is $min(800, n_o)$. Let n_1 pairs of the form (r_1 *tuple, pointer to* r_2) fit in one disk block. Therefore the intermediate result after replacing the r_1 pointers will occupy $B_{o2} = \lceil (n_o/n_1) \rceil$ blocks.

Hence the first pass of replacing the r_1-pointers will cost $B_f = B_{s1} + B_{o1} + min(800, n_o) + B_{o2}$ disk accesses.

The second pass for replacing the r_2-pointers has a similar analysis. Let n_2 tuples of the final join fit in one block. Then the second pass of replacing the r_2-pointers will cost $B_s = B_{s2} + B_{o2} + min(1500, n_o)$ disk accesses, where $B_{s2} = B_{o2}(2\lceil log_{M-1}(B_{o2}/M)\rceil + 2)$.

Hence the total number of disk accesses for the join is $B_j + B_f + B_s$, and the number of pages of output is $\lceil n_o/n_2 \rceil$.

13.9 Let r and s be relations with no indices, and assume that the relations are not sorted. Assuming infinite memory, what is the lowest cost way (in terms of I/O operations) to compute $r \bowtie s$? What is the amount of memory required for this algorithm?

Answer: We can store the entire smaller relation in memory, read the larger relation block by block and perform nested loop join using the larger one as the outer relation. The number of I/O operations is equal to $b_r + b_s$, and memory requirement is $min(b_r, b_s) + 2$ pages.

13.10 Suppose that a B$^+$-tree index on *branch-city* is available on relation *branch*, and that no other index is available. List different ways to handle the following selections that involve negation?

a. $\sigma_{\neg(branch\text{-}city < \text{“Brooklyn”})}(branch)$
b. $\sigma_{\neg(branch\text{-}city = \text{“Brooklyn”})}(branch)$
c. $\sigma_{\neg(branch\text{-}city < \text{“Brooklyn”} \vee assets < 5000)}(branch)$

Answer:

a. Use the index to locate the first tuple whose *branch-city* field has value “Brooklyn”. From this tuple, follow the pointer chains till the end, retrieving all the tuples.
b. For this query, the index serves no purpose. We can scan the file sequentially and select all tuples whose *branch-city* field is anything other than “Brooklyn”.
c. This query is equivalent to the query

$$\sigma_{(branch\text{-}city \geq \text{“Brooklyn”} \wedge assets < 5000)}(branch)$$

Using the *branch-city* index, we can retrieve all tuples with *branch-city* value greater than or equal to “Brooklyn” by following the pointer chains from the first “Brooklyn” tuple. We also apply the additional criteria of $assets < 5000$ on every tuple.

13.11 The hash join algorithm as described in Section 13.5.5 computes the natural join of two relations. Describe how to extend the hash join algorithm to compute the natural left outer join, the natural right outer join and the natural full outer join. (Hint: Keep extra information with each tuple in the hash index, to detect whether any tuple in the probe relation matches the tuple in the hash index.) Try out your algorithm on the *customer* and *depositor* relations.

customer-name	*customer-street*	*customer-city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Curry	North	Rye
Smith	North	Rye
Turner	Putnam	Stamford
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Williams	Nassau	Princeton

Figure 13.17 Sample *customer* relation

Answer: For the probe relation tuple t_r under consideration, if no matching tuple is found in the build relation's hash partition, it is padded with nulls and included in the result. This will give us the natural left outer join $t_r \ \overline{\bowtie} \ t_s$. To get the natural right outer join $t_r \ \overline{\bowtie} \ t_s$, we can keep a boolean flag with each tuple in the current build relation partition H_{s_i} residing in memory, and set it whenever any probe relation tuple matches with it. When we are finished with H_{s_i}, all the tuples in it which do not have their flag set, are padded with nulls and included in the result. To get the natural full outer join, we do both the above operations together.

To try out our algorithm, we use the sample *customer* and *depositor* relations of Figures 13.17 and 13.18. Let us assume that there is enough memory to hold three tuples of the build relation plus a hash index for those three tuples. We use *depositor* as the build relation. We use the simple hashing function which returns the first letter of *customer-name*. Taking the first partitions, we get $H_{r_1} = \{$("Adams", "Spring", "Pittsfield")$\}$, and $H_{s_1} = \phi$. The tuple in the probe relation partition will have no matching tuple, so ("Adams", "Spring", "Pittsfield", $null$) is outputted. In the partition for "D", the lone build relation tuple is unmatched, thus giving an output tuple ("David", $null$, $null$, A-306). In the partition for "H", we find a match for the first time, producing the output tuple ("Hayes", "Main", "Harrison", A-102). Proceeding in a similar way, we process all the partitions and complete the join.

13.12 Write pseudocode for an iterator that implements indexed nested-loop join, where the outer relation is pipelined. Use the standard iterator functions in your pseudocode. Show what state information the iterator must maintain between calls.

Answer: Let *outer* be the iterator which returns successive tuples from the pipelined outer relation. Let *inner* be the iterator which returns successive tuples of the inner relation having a given value at the join attributes. The *inner* iterator returns these tuples by performing an index lookup. The functions **In-**

customer-name	*account-number*
Johnson	A-101
Johnson	A-201
Jones	A-217
Smith	A-215
Hayes	A-102
Turner	A-305
David	A-306
Lindsay	A-222

Figure 13.18 Sample *depositor* relation

dexedNLJoin::open, **IndexedNLJoin::close** and **IndexedNLJoin::next** to implement the indexed nested-loop join iterator are given below. The two iterators *outer* and *inner*, the value of the last read outer relation tuple t_r and a flag $done_r$ indicating whether the end of the outer relation scan has been reached are the state information which need to be remembered by **IndexedNLJoin** between calls.

```
IndexedNLJoin::open()
begin
    outer.open();
    inner.open();
    done_r := false;
    if(outer.next() ≠ false)
        move tuple from outer's output buffer to t_r;
    else
        done_r := true;
end
```

```
IndexedNLJoin::close()
begin
    outer.close();
    inner.close();
end
```

```
boolean IndexedNLJoin::next()
begin
    while(¬done_r)
    begin
        if(inner.next(t_r[JoinAttrs]) ≠ false)
        begin
            move tuple from inner's output buffer to t_s;
            compute t_r ⋈ t_s and place it in output buffer;
            return true;
        end
        else
            if(outer.next() ≠ false)
            begin
                move tuple from outer's output buffer to t_r;
                rewind inner to first tuple of s;
            end
            else
                done_r := true;
    end
    return false;
end
```

13.13 Design sorting based and hashing algorithms for computing the division operation.

Answer: Suppose $r(T \cup S)$ and $s(S)$ be two relations and $r \div s$ has to be computed.

For sorting based algorithm, sort relation s on S. Sort relation r on (T, S). Now, start scanning r and look at the T attribute values of the first tuple. Scan r till tuples have same value of T. Also scan s simultaneously and check whether every tuple of s also occurs as the S attribute of r, in a fashion similar to merge join. If this is the case, output that value of T and proceed with the next value of T. Relation s may have to be scanned multiple times but r will only be scanned once. Total disk accesses, after sorting both the relations, will be $|r| + N * |s|$, where N is the number of distinct values of T in r.

We assume that for any value of T, all tuples in r with that T value fit in memory, and consider the general case at the end. Partition the relation r on attributes in T such that each partition fits in memory (always possible because of our assumption). Consider partitions one at a time. Build a hash table on the tuples, at the same time collecting all distinct T values in a separate hash table. For each value of T, Now, for each value V_T of T, each value s of S, probe the hash table on (V_T, s). If any of the values is absent, discard the value V_T, else output the value V_T.

In the case that not all r tuples with one value for T fit in memory, partition r and s on the S attributes such that the condition is satisfied, run the algorithm

on each corresponding pair of partitions r_i and s_i. Output the intersection of the T values generated in each partition.

CHAPTER 14

Query Optimization

This chapter describes how queries are optimized. It starts off with statistics used for query optimization, and outlines how to use these statistics to estimate selectivities and query result sizes used for cost estimation. Equivalence rules are covered next, followed by a description of a query optimization algorithm modeled after the classic System R optimization algorithm, and coverage of nested subquery optimization. The chapter ends with a description of materialized views, their role in optimization and a description of incremental view-maintenance algorithms.

It should be emphasized that the estimates of query sizes and selectivities are approximate, even if the assumptions made, such as uniformity, hold. Further, the cost estimates for various algorithms presented in Chapter 13 assume only a minimal amount of memory, and are thus worst case estimates with respect to buffer space availability. As a result, cost estimates are never very accurate. However, practical experience has shown that such estimates tend to be reasonably accurate, an plans optimal with respect to estimated cost are rarely much worse than a truly optimal plan.

We do *not* expect students to memorize the size-estimates, and we stress only the process of arriving at the estimates, not the exact values. Precision in terms of estimated cost is not a worthwhile goal, so estimates off by a few I/O operations can be considered acceptable.

If a commercial database system is available for student use, a lab assignment may be designed in which students measure the performance speedup provided by indices. Many commercial database products have an "explain plan" feature that lets the user find the evaluation plan used on a query. It is worthwhile asking students to explore the plans generated for different queries, with and without indices. A more challenging assignment is to design tests to see how clever the query optimizer is, and to guess from these experiments which of the optimization techniques covered in the chapter are used in the system.

Changes from 3rd edition:
The major change from the previous edition is that the 3rd edition chapter on query processing has been split into two chapters.

Coverage of size estimation for different operations, which was earlier covered along with algorithms for the operations has now been separated out into a separate section (Section 14.2). Some of the formulae for estimation of statistics have been simplified and a few new ones have been added.

Pseudocode has been provided for the dynamic programming algorithm for join order optimization. There is a new section on optimization of nested subqueries, which forms an important part of SQL optimization. The section on materialized views is also new to this edition.

Exercises

14.1 Clustering indices may allow faster access to data than a nonclustering index affords. When must we create a nonclustering index, despite the advantages of a clustering index? Explain your answer.
Answer: There can be only one clustering index for a file, based on the ordering key. Any query which needs to search on the other non-ordering keys will need the non-clustering index. If the query accesses a majority of the tuples in the file, it may be more efficient to sort the file on the desired key, rather than using the non-clustering index.

14.2 Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$, with primary keys *A*, *C*, and *E*, respectively. Assume that r_1 has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $r_1 \bowtie r_2 \bowtie r_3$, and give an efficient strategy for computing the join.
Answer:

- The relation resulting from the join of r_1, r_2, and r_3 will be the same no matter which way we join them, due to the associative and commutative properties of joins. So we will consider the size based on the strategy of $((r_1 \bowtie r_2) \bowtie r_3)$. Joining r_1 with r_2 will yield a relation of at most 1000 tuples, since C is a key for r_2. Likewise, joining that result with r_3 will yield a relation of at most 1000 tuples because *E* is a key for r_3. Therefore the final relation will have at most 1000 tuples.
- An efficient strategy for computing this join would be to create an index on attribute *C* for relation r_2 and on *E* for r_3. Then for each tuple in r_1, we do the following:
  - **a.** Use the index for r_2 to look up at most one tuple which matches the *C* value of r_1.
  - **b.** Use the created index on E to look up in r_3 at most one tuple which matches the unique value for E in r_2.

14.3 Consider the relations $r_1(A, B, C)$, $r_2(C, D, E)$, and $r_3(E, F)$ of Exercise 14.2. Assume that there are no primary keys, except the entire schema. Let $V(C, r_1)$ be 900, $V(C, r_2)$ be 1100, $V(E, r_2)$ be 50, and $V(E, r_3)$ be 100. Assume that r_1

has 1000 tuples, r_2 has 1500 tuples, and r_3 has 750 tuples. Estimate the size of $r_1 \bowtie r_2 \bowtie r_3$, and give an efficient strategy for computing the join.
Answer: The estimated size of the relation can be determined by calculating the average number of tuples which would be joined with each tuple of the second relation. In this case, for each tuple in r_1, $1500/V(C, r_2)$ = 15/11 tuples (on the average) of r_2 would join with it. The intermediate relation would have 15000/11 tuples. This relation is joined with r_3 to yield a result of approximately 10,227 tuples (15000/11 $\times$ 750/100 = 10227). A good strategy should join r_1 and r_2 first, since the intermediate relation is about the same size as r_1 or r_2. Then r_3 is joined to this result.

14.4 Suppose that a B$^+$-tree index on *branch-city* is available on relation *branch*, and that no other index is available. What would be the best way to handle the following selections that involve negation?

a. $\sigma_{\neg(branch\text{-}city < \text{“Brooklyn”})}(branch)$
b. $\sigma_{\neg(branch\text{-}city = \text{“Brooklyn”})}(branch)$
c. $\sigma_{\neg(branch\text{-}city < \text{“Brooklyn”} \vee assets < 5000)}(branch)$

Answer:

a. Use the index to locate the first tuple whose *branch-city* field has value “Brooklyn”. From this tuple, follow the pointer chains till the end, retrieving all the tuples.
b. For this query, the index serves no purpose. We can scan the file sequentially and select all tuples whose *branch-city* field is anything other than “Brooklyn”.
c. This query is equivalent to the query $\sigma_{(branch\text{-}city \geq \text{“Brooklyn”} \wedge assets < 5000)}(branch)$. Using the *branch-city* index, we can retrieve all tuples with *branch-city* value greater than or equal to “Brooklyn” by following the pointer chains from the first “Brooklyn” tuple. We also apply the additional criteria of $assets < 5000$ on every tuple.

14.5 Suppose that a B$^+$-tree index on (*branch-name, branch-city*) is available on relation *branch*. What would be the best way to handle the following selection?

$$\sigma_{(branch\text{-}city < \text{“Brooklyn”}) \wedge (assets < 5000) \wedge (branch\text{-}name = \text{“Downtown”})}(branch)$$

Answer: Using the index, we locate the first tuple having *branch-name* “Downtown”. We then follow the pointers retrieving successive tuples as long as *branch-city* is less than “Brooklyn”. From the tuples retrieved, the ones not satisfying the condition $(assets < 5000)$ are rejected.

14.6 Show that the following equivalences hold. Explain how you can apply then to improve the efficiency of certain queries:

a. $E_1 \bowtie_\theta (E_2 - E_3) = (E_1 \bowtie_\theta E_2 - E_1 \bowtie_\theta E_3)$.
b. $\sigma_\theta({}_A\mathcal{G}_F(E)) = {}_A\mathcal{G}_F(\sigma_\theta(E))$, where θ uses only attributes from A.
c. $\sigma_\theta(E_1 \supset\!\bowtie E_2) = \sigma_\theta(E_1) \supset\!\bowtie E_2$ where θ uses only attributes from E_1.

Answer:

a. $E_1 \bowtie_\theta (E_2 - E_3) = (E_1 \bowtie_\theta E_2 - E_1 \bowtie_\theta E_3)$.

Let us rename $(E_1 \bowtie_\theta (E_2 - E_3))$ as R_1, $(E_1 \bowtie_\theta E_2)$ as R_2 and $(E_1 \bowtie_\theta E_3)$ as R_3. It is clear that if a tuple t belongs to R_1, it will also belong to R_2. If a tuple t belongs to R_3, $t[E_3$'s attributes] will belong to E_3, hence t cannot belong to R_1. From these two we can say that

$$\forall t,\ t \in R_1 \ \Rightarrow\ t \in (R_2 - R_3)$$

It is clear that if a tuple t belongs to $R_2 - R_3$, then $t[R_2$'s attributes$] \in E_2$ and $t[R_2$'s attributes$] \notin E_3$. Therefore:

$$\forall t,\ t \in (R_2 - R_3) \ \Rightarrow\ t \in R_1$$

The above two equations imply the given equivalence.

This equivalence is helpful because evaluation of the right hand side join will produce many tuples which will finally be removed from the result. The left hand side expression can be evaluated more efficiently.

b. $\sigma_\theta({}_A\mathcal{G}_F(E)) = {}_A\mathcal{G}_F(\sigma_\theta(E))$, where θ uses only attributes from A.

θ uses only attributes from A. Therefore if any tuple t in the output of ${}_A\mathcal{G}_F(E)$ is filtered out by the selection of the left hand side, all the tuples in E whose value in A is equal to $t[A]$ are filtered out by the selection of the right hand side. Therefore:

$$\forall t,\ t \notin \sigma_\theta({}_A\mathcal{G}_F(E)) \ \Rightarrow\ t \notin {}_A\mathcal{G}_F(\sigma_\theta(E))$$

Using similar reasoning, we can also conclude that

$$\forall t,\ t \notin {}_A\mathcal{G}_F(\sigma_\theta(E)) \ \Rightarrow\ t \notin \sigma_\theta({}_A\mathcal{G}_F(E))$$

The above two equations imply the given equivalence.

This equivalence is helpful because evaluation of the right hand side avoids performing the aggregation on groups which are anyway going to be removed from the result. Thus the right hand side expression can be evaluated more efficiently than the left hand side expression.

c. $\sigma_\theta(E_1 \supset\!\bowtie E_2) = \sigma_\theta(E_1) \supset\!\bowtie E_2$ where θ uses only attributes from E_1.

θ uses only attributes from E_1. Therefore if any tuple t in the output of $(E_1 \supset\!\bowtie E_2)$ is filtered out by the selection of the left hand side, all the tuples in E_1 whose value is equal to $t[E_1]$ are filtered out by the selection of the right hand side. Therefore:

$$\forall t,\ t \notin \sigma_\theta(E_1 \supset\!\bowtie E_2) \ \Rightarrow\ t \notin \sigma_\theta(E_1) \supset\!\bowtie E_2$$

Using similar reasoning, we can also conclude that

$$\forall t,\ t \notin \sigma_\theta(E_1) \supset\!\bowtie E_2 \ \Rightarrow\ t \notin \sigma_\theta(E_1 \supset\!\bowtie E_2)$$

The above two equations imply the given equivalence.

This equivalence is helpful because evaluation of the right hand side avoids producing many output tuples which are anyway going to be re-

moved from the result. Thus the right hand side expression can be evaluated more efficiently than the left hand side expression.

14.7 Show how to derive the following equivalences by a sequence of transformations using the equivalence rules in Section 14.3.1.

a. $\sigma_{\theta_1 \wedge \theta_2 \wedge \theta_3}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(E)))$

b. $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta_3} E_2) = \sigma_{\theta_1}(E_1 \bowtie_{\theta_3} (\sigma_{\theta_2}(E_2)))$, where θ_2 involves only attributes from E_2

Answer:

a. Using rule 1, $\sigma_{\theta_1 \wedge \theta_2 \wedge \theta_3}(E)$ becomes $\sigma_{\theta_1}(\sigma_{\theta_2 \wedge \theta_3}(E))$. On applying rule 1 again, we get $\sigma_{\theta_1}(\sigma_{\theta_2}(\sigma_{\theta_3}(E)))$.

b. $\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_{\theta_3} E_2)$ on applying rule 1 becomes $\sigma_{\theta_1}(\sigma_{\theta_2}(E_1 \bowtie_{\theta_3} E_2))$. This on applying rule 7.a becomes $\sigma_{\theta_1}(E_1 \bowtie_{\theta_3} (\sigma_{\theta_2}(E_2)))$.

14.8 For each of the following pairs of expressions, give instances of relations that show the expressions are not equivalent.

a. $\Pi_A(R - S)$ and $\Pi_A(R) - \Pi_A(S)$

b. $\sigma_{B<4}(\ {}_A\mathcal{G}_{max(B)}(R))$ and ${}_A\mathcal{G}_{max(B)}(\sigma_{B<4}(R))$

c. In the preceding expressions, if both occurrences of max were replaced by min would the expressions be equivalent?

d. $(R \mathbin{⟕} S) \mathbin{⟕} T$ and $R \mathbin{⟕} (S \mathbin{⟕} T)$
In other words, the natural left outer join is not associative.
(Hint: Assume that the schemas of the three relations are $R(a, b1)$, $S(a, b2)$, and $T(a, b3)$, respectively.)

e. $\sigma_\theta(E_1 \mathbin{⟕} E_2)$ and $E_1 \mathbin{⟕} \sigma_\theta(E_2)$, where θ uses only attributes from E_2

Answer:

a. $R = \{(1, 2)\},\ S = \{(1, 3)\}$
The result of the left hand side expression is $\{(1)\}$, whereas the result of the right hand side expression is empty.

b. $R = \{(1, 2), (1, 5)\}$
The left hand side expression has an empty result, whereas the right hand side one has the result $\{(1, 2)\}$.

c. Yes, on replacing the max by the min, the expressions will become equivalent. Any tuple that the selection in the rhs eliminates would not pass the selection on the lhs if it were the minimum value, and would be eliminated anyway if it were not the minimum value.

d. $R = \{(1, 2)\},\ S = \{(2, 3)\},\ T = \{(1, 4)\}$. The left hand expression gives $\{(1, 2, null, 4)\}$ whereas the the right hand expression gives $\{(1, 2, 3, null)\}$.

e. Let R be of the schema (A, B) and S of (A, C). Let $R = \{(1, 2)\},\ S = \{(2, 3)\}$ and let θ be the expression $C = 1$. The left side expression's result is empty, whereas the right side expression results in $\{(1, 2, null)\}$.

14.9 SQL allows relations with duplicates (Chapter 4).

a. Define versions of the basic relational-algebra operations σ, Π, $\times$, $\bowtie$, $-$, $\cup$, and $\cap$ that work on relations with duplicates, in a way consistent with SQL.

b. Check which of the equivalence rules 1 through 7.b hold for the multiset version of the relational-algebra defined in part a.

Answer:

a. We define the multiset versions of the relational-algebra operators here. Given multiset relations r_1 and r_2,

i. σ

Let there be c_1 copies of tuple t_1 in r_1. If t_1 satisfies the selection σ_θ, then there are c_1 copies of t_1 in $\sigma_\theta(r_1)$, otherwise there are none.

ii. Π

For each copy of tuple t_1 in r_1, there is a copy of tuple $\Pi_A(t_1)$ in $\Pi_A(r_1)$, where $\Pi_A(t_1)$ denotes the projection of the single tuple t_1.

iii. $\times$

If there are c_1 copies of tuple t_1 in r_1 and c_2 copies of tuple t_2 in r_2, then there are $c_1 * c_2$ copies of the tuple $t_1.t_2$ in $r_1 \times r_2$.

iv. $\bowtie$

The output will be the same as a cross product followed by a selection.

v. $-$

If there are c_1 copies of tuple t in r_1 and c_2 copies of t in r_2, then there will be $c_1 - c_2$ copies of t in $r_1 - r_2$, provided that $c_1 - c_2$ is positive.

vi. $\cup$

If there are c_1 copies of tuple t in r_1 and c_2 copies of t in r_2, then there will be $c_1 + c_2$ copies of t in $r_1 \cup r_2$.

vii. $\cap$

If there are c_1 copies of tuple t in r_1 and c_2 copies of t in r_2, then there will be $min(c_1, c_2)$ copies of t in $r_1 \cap r_2$.

b. All the equivalence rules 1 through 7.b of section 14.3.1 hold for the multiset version of the relational-algebra defined in the first part.

There exist equivalence rules which hold for the ordinary relational-algebra, but do not hold for the multiset version. For example consider the rule :-

$$A \cap B \quad = \quad A \cup B \; - \; (A - B) \; - \; (B - A)$$

This is clearly valid in plain relational-algebra. Consider a multiset instance in which a tuple t occurs 4 times in A and 3 times in B. t will occur 3 times in the output of the left hand side expression, but 6 times in the output of the right hand side expression. The reason for this rule to not hold in the multiset version is the asymmetry in the semantics of multiset union and intersection.

14.10 $**$Show that, with n relations, there are $(2(n-1))!/(n-1)!$ different join orders.

Hint: A **complete binary tree** is one where every internal node has exactly two children. Use the fact that the number of different complete binary trees with n leaf nodes is $\frac{1}{n}\binom{2(n-1)}{(n-1)}$.

If you wish, you can derive the formula for the number of complete binary trees with n nodes from the formula for the number of binary trees with n nodes. The number of binary trees with n nodes is $\frac{1}{n+1}\binom{2n}{n}$; this number is known as the Catalan number, and its derivation can be found in any standard textbook on data structures or algorithms.

Answer: Each join order is a complete binary tree (every non-leaf node has exactly two children) with the relations as the leaves. The number of different complete binary trees with n leaf nodes is $\frac{1}{n}\binom{2(n-1)}{(n-1)}$. This is because there is a bijection between the number of complete binary trees with n leaves and number of binary trees with $n-1$ nodes. Any complete binary tree with n leaves has $n-1$ internal nodes. Removing all the leaf nodes, we get a binary tree with $n-1$ nodes. Conversely, given any binary tree with $n-1$ nodes, it can be converted to a complete binary tree by adding n leaves in a unique way. The number of binary trees with $n-1$ nodes is given by $\frac{1}{n}\binom{2(n-1)}{(n-1)}$, known as the Catalan number. Multiplying this by $n!$ for the number of permutations of the n leaves, we get the desired result.

14.11 $**$Show that the lowest-cost join order can be computed in time $O(3^n)$. Assume that you can store and look up information about a set of relations (such as the optimal join order for the set, and the cost of that join order) in constant time. (If you find this exercise difficult, at least show the looser time bound of $O(2^{2n})$.)

Answer: Consider the dynamic programming algorithm given in Section 14.4.2. For each subset having $k+1$ relations, the optimal join order can be computed in time 2^{k+1}. That is because for one particular pair of subsets A and B, we need constant time and there are at most $2^{k+1}-2$ different subsets that A can denote. Thus, over all the $\binom{n}{k+1}$ subsets of size $k+1$, this cost is $\binom{n}{k+1}2^{k+1}$. Summing over all k from 1 to $n-1$ gives the binomial expansion of $((1+x)^n - x)$ with $x=2$. Thus the total cost is less than 3^n.

14.12 Show that, if only left-deep join trees are considered, as in the System R optimizer, the time taken to find the most efficient join order is around $n2^n$. Assume that there is only one interesting sort order.

Answer: The derivation of time taken is similar to the general case, except that instead of considering $2^{k+1}-2$ subsets of size less than or equal to k for A, we only need to consider $k+1$ subsets of size exactly equal to k. That is because the right hand operand of the topmost join has to be a single relation. Therefore the total cost for finding the best join order for all subsets of size $k+1$ is $\binom{n}{k+1}(k+1)$, which is equal to $n\binom{n-1}{k}$. Summing over all k from 1 to $n-1$ using the binomial expansion of $(1+x)^{n-1}$ with $x=1$, gives a total cost of less than $n2^{n-1}$.

14.13 A set of equivalence rules is said to be *complete* if, whenever two expressions are equivalent, one can be derived from the other by a sequence of uses of the equivalence rules. Is the set of equivalence rules that we considered in Section 14.3.1 complete? Hint: Consider the equivalence $\sigma_{3=5}(r) = \{\ \}$.

Answer: Two relational expressions are defined to be *equivalent* when on all input relations, they give the same output. The set of equivalence rules considered in Section 14.3.1 is not complete. The expressions $\sigma_{3=5}(r)$ and $\{\ \}$ are equivalent, but this cannot be shown by using just these rules.

14.14 Decorrelation:

a. Write a nested query on the relation *account* to find for each branch with name starting with "B", all accounts with the maximum balance at the branch.

b. Rewrite the preceding query, without using a nested subquery; in other words, decorrelate the query.

c. Give a procedure (similar that that described in Section 14.4.5) for decorrelating such queries.

Answer:

a. The nested query is as follows:

```
select  S.acount-number
from    account S
where   S.branch-name like 'B%' and
        S.balance =
        (select max(T.balance)
        from account T
        where T.branch-name = S.branch-name)
```

b. The decorrelated query is as follows:

```
create table t1 as
        select branch-name, max(balance)
        from account
        group by branch-name
select  account-number
from    account, t1
where   account.branch-name like 'B%' and
        account.branch-name = t1.branch-name and
        account.balance = t1.balance
```

c. In general, consider the queries of the form:

select $\cdots$
from L_1
where P_1 **and**
A_1 **op**
(**select f**(A_2)
from L_2
where P_2)

where, f is some aggregate function on attributes A_2, and *op* is some boolean binary operator. It can be rewritten as

create table t_1 **as**
select f(A_2),V
from L_2
where P_2^1
group by V
select $\cdots$
from L_1, t_1
where P_1 **and** P_2^2 **and**
A_1 **op** $t_1.A_2$

where P_2^1 contains predicates in P_2 without selections involving correlation variables, and P_2^2 introduces the selections involving the correlation variables. V contains all the attributes that are used in the selections involving correlation variables in the nested query.

14.15 Describe how to incrementally maintain the results of the following operations, on both insertions and deletions.

a. Union and set difference
b. Left outer join

Answer:

a. Given materialized view $v = r \cup s$, when a tuple is inserted in r, we check if it is present in v, and if not we add it to v. When a tuple is deleted from r, we check if it is there in s, and if not, we delete it from v. Inserts and deletes in s are handled in symmetric fashion.

For set difference, given view $v = r - s$, when a tuple is inserted in r, we check if it is present in s, and if not we add it to v. When a tuple is deleted from r, we delete it from v if present. When a tuple is inserted in s, we delete it from v if present. When a tuple is deleted from s, we check if it is present in r, and if so we add it to v.

b. Given materialized view $v = r \supset\!\bowtie s$, when a set of tuples i_r is inserted in r, we add the tuples $i_r \supset\!\bowtie s$ to the view. When i_r is deleted from r, we delete $i_r \supset\!\bowtie s$ from the view. When a set of tuples i_s is inserted in s, we compute $r \bowtie i_s$. We find all the tuples of r which previously did not match any tuple from s(i.e. those padded with *null* in $r \supset\!\bowtie s$) but which match i_s. We remove all those *null* padded entries from the view and add the tuples

$r \bowtie s$ to the view. When i_s is deleted from s, we delete the tuples $r \bowtie i_s$ from the view. Also we find all the tuples in r which match i_s but which do not match any other tuples in s. We add all those to the view, after padding them with *null* values.

14.16 Give an example of an expression defining a materialized view and two situations (sets of statistics for the input relations and the differentials) such that incremental view maintenance is better than recomputation in one situation, and recomputation is better in the other situation.

Answer: Let r and s be two relations. Consider a materialized view on these defined by $(r \bowtie s)$. Suppose 70% of the tuples in r are deleted. Then recompution is better that incremental view maintenance. This is because in incremental view maintenance, the 70% of the deleted tuples have to be joined with s while in recompution, just the remaining 30% of the tuples in r have to be joined with s.

However, if the number of tuples in r is increased by a small percentage, for example 2%, then incremental view maintenance is likely to be better than recomputation.

CHAPTER 15

Transactions

This chapter provides an overview of transaction processing. It first motivates the problems of atomicity, consistency, isolation and durability, and introduces the notion of ACID transactions. It then presents some naive schemes, and their drawbacks, thereby motivating the techniques described in Chapters 16 and 17. The rest of the chapter describes the notion of schedules and the concept of serializability.

We strongly recommend covering this chapter in a first course on databases, since it introduces concepts that every database student should be aware of. Details on how to implement the transaction properties are covered in Chapters 16 and 17.

In the initial presentation to the ACID requirements, the isolation requirement on concurrent transactions does not insist on serializability. Following Haerder and Reuter [1983], isolation just requires that the events within a transaction must be hidden from other transactions running concurrently, in order to allow rollback. However, later in the chapter, and in most of the book (except in Chapter 24), we use the stronger condition of serializability as a requirement on concurrent transactions.

Changes from 3rd edition:
Testing of view serializability has been dropped from this chapter (and from the book), since it does not have any practical significance.

Exercises

15.1 List the ACID properties. Explain the usefulness of each.

Answer: The ACID properties, and the need for each of them are:-

- **Consistency:**
  Execution of a transaction in isolation (that is, with no other transaction executing concurrently) preserves the consistency of the database. This is typically the responsibility of the application programmer who codes the transactions.

- **Atomicity:**
  Either all operations of the transaction are reflected properly in the database, or none are. Clearly lack of atomicity will lead to inconsistency in the database.
- **Isolation:**
  When multiple transactions execute concurrently, it should be the case that, for every pair of transactions T_i and T_j, it appears to T_i that either T_j finished execution before T_i started, or T_j started execution after T_i finished. Thus, each transaction is unaware of other transactions executing concurrently with it. The user view of a transaction system requires the isolation property, and the property that concurrent schedules take the system from one consistent state to another. These requirements are satisfied by ensuring that only serializable schedules of individually consistency preserving transactions are allowed.
- **Durability:**
  After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

15.2 Suppose that there is a database system that never fails. Is a recovery manager required for this system?
Answer: Even in this case the recovery manager is needed to perform roll-back of aborted transactions.

15.3 Consider a file system such as the one on your favorite operating system.

a. What are the steps involved in creation and deletion of files, and in writing data to a file?
b. Explain how the issues of atomicity and durability are relevant to the creation and deletion of files, and to writing data to files.

Answer: There are several steps in the creation of a file. A storage area is assigned to the file in the file system, a unique i-number is given to the file and an i-node entry is inserted into the i-list. Deletion of file involves exactly opposite steps.

For the file system user in UNIX, durability is important for obvious reasons, but atomicity is not relevant generally as the file system doesn't support transactions. To the file system implementor though, many of the internal file system actions need to have transaction semantics. All the steps involved in creation/deletion of the file must be atomic, otherwise there will be unreferenceable files or unusable areas in the file system.

15.4 Database-system implementers have paid much more attention to the ACID properties than have file-system implementers. Why might this be the case?
Answer: Database systems usually perform crucial tasks whose effects need to be atomic and durable, and whose outcome affects the real world in a permanent manner. Examples of such tasks are monetary transactions, seat bookings etc. Hence the ACID properties have to be ensured. In contrast, most users of

file systems would not be willing to pay the price (monetary, disk space, time) of supporting ACID properties.

15.5 During its execution, a transaction passes through several states, until it finally commits or aborts. List all possible sequences of states through which a transaction may pass. Explain why each state transition may occur.
Answer: The possible sequences of states are:-

a. *active → partially committed → committed*. This is the normal sequence a successful transaction will follow. After executing all its statements it enters the *partially committed* state. After enough recovery information has been written to disk, the transaction finally enters the *committed* state.

b. *active → partially committed → aborted*. After executing the last statement of the transaction, it enters the *partially committed* state. But before enough recovery information is written to disk, a hardware failure may occur destroying the memory contents. In this case the changes which it made to the database are undone, and the transaction enters the *aborted* state.

c. *active → failed → aborted*. After the transaction starts, if it is discovered at some point that normal execution cannot continue (either due to internal program errors or external errors), it enters the failed state. It is then rolled back, after which it enters the *aborted* state.

15.6 Justify the following statement: Concurrent execution of transactions is more important when data must be fetched from (slow) disk or when transactions are long, and is less important when data is in memory and transactions are very short.
Answer: If a transaction is very long or when it fetches data from a slow disk, it takes a long time to complete. In absence of concurrency, other transactions will have to wait for longer period of time. Average responce time will increase. Also when the transaction is reading data from disk, CPU is idle. So resources are not properly utilized. Hence concurrent execution becomes important in this case. However, when the transactions are short or the data is available in memory, these problems do not occur.

15.7 Explain the distinction between the terms *serial schedule* and *serializable schedule*.
Answer: A schedule in which all the instructions belonging to one single transaction appear together is called a *serial schedule*. A *serializable schedule* has a weaker restriction that it should be *equivalent* to some serial schedule. There are two definitions of schedule equivalence – conflict equivalence and view equivalence. Both of these are described in the chapter.

15.8 Consider the following two transactions:

T_1: read(A);
read(B);
if $A = 0$ **then** $B := B + 1$;
write(B).
T_2: read(B);
read(A);
if $B = 0$ **then** $A := A + 1$;
write(A).

Let the consistency requirement be $A = 0 \vee B = 0$, with $A = B = 0$ the initial values.

a. Show that every serial execution involving these two transactions preserves the consistency of the database.
b. Show a concurrent execution of T_1 and T_2 that produces a nonserializable schedule.
c. Is there a concurrent execution of T_1 and T_2 that produces a serializable schedule?

Answer:

a. There are two possible executions: $T_1\ T_2$ and $T_2\ T_1$.

Case 1:

	A	B
initially	0	0
after T_1	0	1
after T_2	0	1

Consistency met: $A = 0 \vee B = 0 \equiv T \vee F = T$

Case 2:

	A	B
initially	0	0
after T_2	1	0
after T_1	1	0

Consistency met: $A = 0 \vee B = 0 \equiv F \vee T = T$

b. Any interleaving of T_1 and T_2 results in a non-serializable schedule.

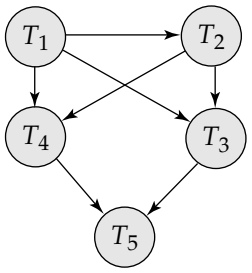


Figure 15.18. Precedence graph.

T_1	T_2
read(A)	
	read(B)
	read(A)
read(B)	
if $A = 0$ **then** $B = B + 1$	
	if $B = 0$ **then** $A = A + 1$
	write(A)
write(B)	

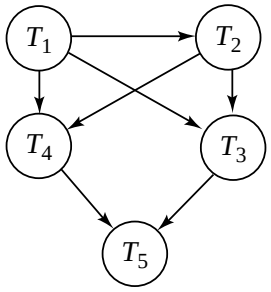
c. There is no parallel execution resulting in a serializable schedule. From part a. we know that a serializable schedule results in $A = 0 \vee B = 0$. Suppose we start with T_1 **read**(A). Then when the schedule ends, no matter when we run the steps of T_2, $B = 1$. Now suppose we start executing T_2 prior to completion of T_1. Then T_2 **read**(B) will give B a value of 0. So when T_2 completes, $A = 1$. Thus $B = 1 \wedge A = 1 \rightarrow \neg (A = 0 \vee B = 0)$. Similarly for starting with T_2 **read**(B).

15.9 Since every conflict-serializable schedule is view serializable, why do we emphasize conflict serializability rather than view serializability?

Answer: Most of the concurrency control protocols (protocols for ensuring that only serializable schedules are generated) used in practise are based on conflict serializability—they actually permit only a subset of conflict serializable schedules. The general form of view serializability is very expensive to test, and only a very restricted form of it is used for concurrency control.

15.10 Consider the precedence graph of Figure 15.18. Is the corresponding schedule conflict serializable? Explain your answer.

Answer: There is a serializable schedule corresponding to the precedence graph below, since the graph is acyclic. A possible schedule is obtained by doing a topological sort, that is, T_1, T_2, T_3, T_4, T_5.



15.11 What is a recoverable schedule? Why is recoverability of schedules desirable? Are there any circumstances under which it would be desirable to allow nonrecoverable schedules? Explain your answer.

Answer: A recoverable schedule is one where, for each pair of transactions T_i and T_j such that T_j reads data items previously written by T_i, the commit

operation of T_i appears before the commit operation of T_j. Recoverable schedules are desirable because failure of a transaction might otherwise bring the system into an irreversibly inconsistent state. Nonrecoverable schedules may sometimes be needed when updates must be made visible early due to time constraints, even if they have not yet been committed, which may be required for very long duration transactions.

15.12 What is a cascadeless schedule? Why is cascadelessness of schedules desirable? Are there any circumstances under which it would be desirable to allow noncascadeless schedules? Explain your answer.

Answer: A cascadeless schedule is one where, for each pair of transactions T_i and T_j such that T_j reads data items previously written by T_i, the commit operation of T_i appears before the read operation of T_j. Cascadeless schedules are desirable because the failure of a transaction does not lead to the aborting of any other transaction. Of course this comes at the cost of less concurrency. If failures occur rarely, so that we can pay the price of cascading aborts for the increased concurrency, noncascadeless schedules might be desirable.

CHAPTER 16

Concurrency Control

This chapter describes how to control concurrent execution in a database, in order to ensure the isolation properties of transactions. A variety of protocols are described for this purpose. If time is short, some of the protocols may be omitted. We recommend covering, at the least, two-phase locking (Sections 16.1.1), through 16.1.3, deadlock detection and recovery (Section 16.6, omitting Section 16.6.1), the phantom phenomenon (Section 16.7.3), and the concepts behind index concurrency control (the introductory part of Section 16.9). The most widely used techniques would thereby be covered.

It is worthwhile pointing out how the graph-based locking protocols generalize simple protocols, such as ordered acquisition of locks, which students may have studied in an operating system course. Although the timestamp protocols by themselves are not widely used, multiversion two-phase locking (Section 16.5.2) is of increasing importance since it allows long read-only transactions to run concurrently with updates.

The phantom phenomenon is often misunderstood by students as showing that two-phase locking is incorrect. It is worth stressing that transactions that scan a relation must read some data to find out what tuples are in the relation; as long as this data is itself locked in a two-phase manner, the phantom phenomenon will not arise.

Changes from 3rd edition:

This chapter has been reorganized from the previous edition. Some of the material from the Concurrency Control chapter of the second edition (Chapter 11), such as schedules and testing for serializability have been moved into Chapter 15 of the third edition. The sections on deadlock handling (Section 16.6) and concurrency in index structures (Section 16.9) have been moved in from Chapter 12 of the second edition (Transaction Processing). The section on multiversion two-phase locking is new.

Exercises

16.1 Show that the two-phase locking protocol ensures conflict serializability, and that transactions can be serialized according to their lock points.
Answer: Suppose two-phase locking does not ensure serializability. Then there exists a set of transactions $T_0,\ T_1\ ...\ T_{n-1}$ which obey 2PL and which produce a nonserializable schedule. A non-serializable schedule implies a cycle in the precedence graph, and we shall show that 2PL cannot produce such cycles. Without loss of generality, assume the following cycle exists in the precedence graph: $T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow ... \rightarrow T_{n-1} \rightarrow T_0$. Let α_i be the time at which T_i obtains its last lock (i.e. T_i's lock point). Then for all transactions such that $T_i \rightarrow T_j$, $\alpha_i\ <\ \alpha_j$. Then for the cycle we have

$$\alpha_0\ <\ \alpha_1\ <\ \alpha_2\ <\ ...\ <\ \alpha_{n-1}\ <\ \alpha_0$$

Since $\alpha_0\ <\ \alpha_0$ is a contradiction, no such cycle can exist. Hence 2PL cannot produce non-serializable schedules. Because of the property that for all transactions such that $T_i \rightarrow T_j$, $\alpha_i\ <\ \alpha_j$, the lock point ordering of the transactions is also a topological sort ordering of the precedence graph. Thus transactions can be serialized according to their lock points.

16.2 Consider the following two transactions:

T_{31}: read(A);
read(B);
if $A\ =\ 0$ **then** B := B + 1;
write(B).

T_{32}: read(B);
read(A);
if $B\ =\ 0$ **then** A := A + 1;
write(A).

Add lock and unlock instructions to transactions T_{31} and T_{32}, so that they observe the two-phase locking protocol. Can the execution of these transactions result in a deadlock?
Answer:

a. Lock and unlock instructions:

T_{31}: **lock-S**(A)
read(A)
lock-X(B)
read(B)
if $A\ =\ 0$
then $B\ :=\ B\ +\ 1$
write(B)
unlock(A)
unlock(B)

T_{32}: **lock-S**(B)
read(B)
lock-X(A)
read(A)
if $B = 0$
then $A := A + 1$
write(A)
unlock(B)
unlock(A)

b. Execution of these transactions can result in deadlock. For example, consider the following partial schedule:

T_{31}	T_{32}
lock-S(A)	
	lock-S(B)
	read(B)
read(A)	
lock-X(B)	
	lock-X(A)

The transactions are now deadlocked.

16.3 What benefit does strict two-phase locking provide? What disadvantages result?
Answer: Because it produces only cascadeless schedules, recovery is very easy. But the set of schedules obtainable is a subset of those obtainable from plain two phase locking, thus concurrency is reduced.

16.4 What benefit does rigorous two-phase locking provide? How does it compare with other forms of two-phase locking?
Answer: Rigorous two-phase locking has the advantages of strict 2PL. In addition it has the property that for two conflicting transactions, their commit order is their serializability order. In some systems users might expect this behavior.

16.5 Most implementations of database systems use strict two-phase locking. Suggest three reasons for the popularity of this protocol.
Answer: It is relatively simple to implement, imposes low rollback overhead because of cascadeless schedules, and usually allows an acceptable level of concurrency.

16.6 Consider a database organized in the form of a rooted tree. Suppose that we insert a dummy vertex between each pair of vertices. Show that, if we follow the tree protocol on the new tree, we get better concurrency than if we follow the tree protocol on the original tree.
Answer: The proof is in Buckley and Silberschatz, "Concurrency Control in Graph Protocols by Using Edge Locks," Proc. ACM SIGACT-SIGMOD Symposium on the Principles of Database Systems, 1984.

16.7 Show by example that there are schedules possible under the tree protocol that are not possible under the two-phase locking protocol, and vice versa.
Answer: Consider the tree-structured database graph given below.

Schedule possible under tree protocol but not under 2PL:

T_1	T_2
lock(A)	
lock(B)	
unlock(A)	
	lock(A)
lock(C)	
unlock(B)	
	lock(B)
	unlock(A)
	unlock(B)
unlock(C)	

Schedule possible under 2PL but not under tree protocol:

T_1	T_2
lock(A)	
	lock(B)
lock(C)	
	unlock(B)
unlock(A)	
unlock(C)	

16.8 Consider the following extension to the tree-locking protocol, which allows both shared and exclusive locks:

- A transaction can be either a read-only transaction, in which case it can request only shared locks, or an update transaction, in which case it can request only exclusive locks.
- Each transaction must follow the rules of the tree protocol. Read-only transactions may lock any data item first, whereas update transactions must lock the root first.

Show that the protocol ensures serializability and deadlock freedom.
Answer: The proof is in Kedem and Silberschatz, "Locking Protocols: From Exclusive to Shared Locks," JACM Vol. 30, 4, 1983.

16.9 Consider the following graph-based locking protocol, which allows only exclusive lock modes, and which operates on data graphs that are in the form of a rooted directed acyclic graph.

- A transaction can lock any vertex first.
- To lock any other vertex, the transaction must be holding a lock on the majority of the parents of that vertex.

Show that the protocol ensures serializability and deadlock freedom.
Answer: The proof is in Kedem and Silberschatz, "Controlling Concurrency Using Locking Protocols," Proc. Annual IEEE Symposium on Foundations of Computer Science, 1979.

16.10 Consider the following graph-based locking protocol that allows only exclusive lock modes, and that operates on data graphs that are in the form of a rooted directed acyclic graph.

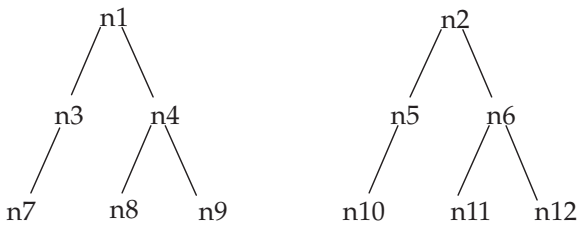
- A transaction can lock any vertex first.
- To lock any other vertex, the transaction must have visited all the parents of that vertex, and must be holding a lock on one of the parents of the vertex.

Show that the protocol ensures serializability and deadlock freedom.
Answer: The proof is in Kedem and Silberschatz, "Controlling Concurrency Using Locking Protocols," Proc. Annual IEEE Symposium on Foundations of Computer Science, 1979.

16.11 Consider a variant of the tree protocol called the *forest* protocol. The database is organized as a forest of rooted trees. Each transaction T_i must follow the following rules:

- The first lock in each tree may be on any data item.
- The second, and all subsequent, locks in a tree may be requested only if the parent of the requested node is currently locked.
- Data items may be unlocked at any time.
- A data item may not be relocked by T_i after it has been unlocked by T_i.

Show that the forest protocol does *not* ensure serializability.
Answer: Take a system with 2 trees:



We have 2 transactions, T_1 and T_2. Consider the following legal schedule:

T_1	T_2
lock(n1)	
lock(n3)	
write(n3)	
unlock(n3)	
	lock(n2)
	lock(n5)
	write(n5)
	unlock(n5)
lock(n5)	
read(n5)	
unlock(n5)	
unlock(n1)	
	lock(n3)
	read(n3)
	unlock(n3)
	unlock(n2)

This schedule is not serializable.

16.12 Locking is not done explicitly in persistent programming languages. Rather, objects (or the corresponding pages) must be locked when the objects are accessed. Most modern operating systems allow the user to set access protections (no access, read, write) on pages, and memory access that violate the access protections result in a protection violation (see the Unix `mprotect` command, for example). Describe how the access-protection mechanism can be used for page-level locking in a persistent programming language. (Hint: The technique is similar to that used for hardware swizzling in Section 11.9.4).

Answer: The access protection mechanism can be used to implement page level locking. Consider reads first. A process is allowed to read a page only after it read-locks the page. This is implemented by using `mprotect` to initially turn off read permissions to all pages, for the process. When the process tries to access an address in a page, a protection violation occurs. The handler associated with protection violation then requests a read lock on the page, and after the lock is acquired, it uses `mprotect` to allow read access to the page by the process, and finally allows the process to continue. Write access is handled similarly.

16.13 Consider a database system that includes an atomic **increment** operation, in addition to the read and write operations. Let V be the value of data item X. The operation

increment(X) by C

sets the value of X to $V + C$ in an atomic step. The value of X is not available to the transaction unless the latter executes a read(X). Figure 16.23 shows a lock-compatibility matrix for three lock modes: share mode, exclusive mode, and incrementation mode.

	S	X	I
S	true	false	false
X	false	false	false
I	false	false	true

Figure 16.23. Lock-compatibility matrix.

a. Show that, if all transactions lock the data that they access in the corresponding mode, then two-phase locking ensures serializability.
b. Show that the inclusion of **increment** mode locks allows for increased concurrency. (Hint: Consider check-clearing transactions in our bank example.)

Answer: The proof is in Korth, "Locking Primitives in a Database System," JACM Vol. 30, 1983.

16.14 In timestamp ordering, **W-timestamp**(Q) denotes the largest timestamp of any transaction that executed write(Q) successfully. Suppose that, instead, we defined it to be the timestamp of the most recent transaction to execute write(Q) successfully. Would this change in wording make any difference? Explain your answer.
Answer: It would make no difference. The write protocol is such that the most recent transaction to write an item is also the one with the largest timestamp to have done so.

16.15 When a transaction is rolled back under timestamp ordering, it is assigned a new timestamp. Why can it not simply keep its old timestamp?
Answer: A transaction is rolled back because a newer transaction has read or written the data which it was supposed to write. If the rolled back transaction is re-introduced with the same timestamp, the same reason for rollback is still valid, and the transaction will have be rolled back again. This will continue indefinitely.

16.16 In multiple-granularity locking, what is the difference between implicit and explicit locking?
Answer: When a transaction *explicitly* locks a node in shared or exclusive mode, it *implicitly* locks all the descendents of that node in the same mode. The transaction need not explicitly lock the descendent nodes. There is no difference in the functionalities of these locks, the only difference is in the way they are acquired, and their presence tested.

16.17 Although SIX mode is useful in multiple-granularity locking, an exclusive and intend-shared (XIS) mode is of no use. Why is it useless?
Answer: An exclusive lock is incompatible with any other lock kind. Once a node is locked in exclusive mode, none of the descendents can be simultaneously accessed by any other transaction in any mode. Therefore an exclusive and intend-shared declaration has no meaning.

16.18 Use of multiple-granularity locking may require more or fewer locks than an equivalent system with a single lock granularity. Provide examples of both situations, and compare the relative amount of concurrency allowed.
Answer: If a transaction needs to access a large a set of items, multiple granularity locking requires fewer locks, whereas if only one item needs to be accessed, the single lock granularity system allows this with just one lock. Because all the desired data items are locked and unlocked together in the multiple granularity scheme, the locking overhead is low, but concurrency is also reduced.

16.19 Consider the validation-based concurrency-control scheme of Section 16.3. Show that by choosing Validation(T_i), rather than Start(T_i), as the timestamp of transaction T_i, we can expect better response time provided that conflict rates among transactions are indeed low.
Answer: In the concurrency control scheme of Section 16.3 choosing **Start**(T_i) as the timestamp of T_i gives a subset of the schedules allowed by choosing **Validation**(T_i) as the timestamp. Using **Start**(T_i) means that whoever started first must finish first. Clearly transactions could enter the validation phase in the same order in which they began executing, but this is overly restrictive. Since choosing **Validation**(T_i) causes fewer nonconflicting transactions to restart, it gives the better response times.

16.20 Show that there are schedules that are possible under the two-phase locking protocol, but are not possible under the timestamp protocol, and vice versa.
Answer: A schedule which is allowed in the two-phase locking protocol but not in the timestamp protocol is:

step	T_0	T_1	Precedence remarks
1	**lock-S**(A)		
2	**read**(A)		
3		**lock-X**(B)	
4		**write**(B)	
5		**unlock**(B)	
6	**lock-S**(B)		
7	**read**(B)		$T_1 \rightarrow T_0$
8	**unlock**(A)		
9	**unlock**(B)		

This schedule is not allowed in the timestamp protocol because at step 7, the W-timestamp of B is 1.

A schedule which is allowed in the timestamp protocol but not in the two-phase locking protocol is:

step	T_0	T_1	T_2
1	**write**(A)		
2		**write**(A)	
3			**write**(A)
4	**write**(B)		
5		**write**(B)	

This schedule cannot have lock instructions added to make it legal under two-phase locking protocol because T_1 must unlock (A) between steps 2 and 3, and must lock (B) between steps 4 and 5.

16.21 For each of the following protocols, describe aspects of practical applications that would lead you to suggest using the protocol, and aspects that would suggest not using the protocol:

- Two-phase locking
- Two-phase locking with multiple-granularity locking
- The tree protocol
- Timestamp ordering
- Validation
- Multiversion timestamp ordering
- Multiversion two-phase locking

Answer:

- Two-phase locking: Use for simple applications where a single granularity is acceptable. If there are large read-only transactions, multiversion protocols would do better. Also, if deadlocks must be avoided at all costs, the tree protocol would be preferable.
- Two-phase locking with multiple granularity locking: Use for an application mix where some applications access individual records and others access whole relations or substantial parts thereof. The drawbacks of 2PL mentioned above also apply to this one.
- The tree protocol: Use if all applications tend to access data items in an order consistent with a particular partial order. This protocol is free of deadlocks, but transactions will often have to lock unwanted nodes in order to access the desired nodes.
- Timestamp ordering: Use if the application demands a concurrent execution that is equivalent to a particular serial ordering (say, the order of arrival), rather than *any* serial ordering. But conflicts are handled by roll-back of transactions rather than waiting, and schedules are not recoverable. To make them recoverable, additional overheads and increased response time have to be tolerated. Not suitable if there are long read-only transactions, since they will starve. Deadlocks are absent.
- Validation: If the probability that two concurrently executing transactions conflict is low, this protocol can be used advantageously to get better concurrency and good response times with low overheads. Not suitable under high contention, when a lot of wasted work will be done.
- Multiversion timestamp ordering: Use if timestamp ordering is appropriate but it is desirable for read requests to never wait. Shares the other disadvantages of the timestamp ordering protocol.
- Multiversion two-phase locking: This protocol allows read-only transactions to always commit without ever waiting. Update transactions follow 2PL, thus allowing recoverable schedules with conflicts solved by waiting

rather than roll-back. But the problem of deadlocks comes back, though read-only transactions cannot get involved in them. Keeping multiple versions adds space and time overheads though, therefore plain 2PL may be preferable in low conflict situations.

16.22 Under a modified version of the timestamp protocol, we require that a commit bit be tested to see whether a read request must wait. Explain how the commit bit can prevent cascading abort. Why is this test not necessary for write requests?

Answer: Using the commit bit, a read request is made to wait if the transaction which wrote the data item has not yet committed. Therefore, if the writing transaction fails before commit, we can abort that transaction alone. The waiting read will then access the earlier version in case of a multiversion system, or the restored value of the data item after abort in case of a single-version system. For writes, this commit bit checking is unnecessary. That is because either the write is a "blind" write and thus independent of the old value of the data item or there was a prior read, in which case the test was already applied.

16.23 Explain why the following technique for transaction execution may provide better performance than just using strict two-phase locking: First execute the transaction without acquiring any locks and without performing any writes to the database as in the validation based techniques, but unlike in the validation techniques do not perform either validation or perform writes on the database. Instead, rerun the transaction using strict two-phase locking. (Hint: Consider waits for disk I/O.)

Answer: A transaction waits on *a.* disk I/Oand *b.* lock acquisition. Transactions generally wait on disk reads and not on disk writes as disk writes are handled by the buffering mechanism in asynchronous fashion and transactions update only the in-memory copy of the disk blocks.

The technique proposed essentially separates the waiting times into two phases. The first phase – where transaction is executed without acquiring any locks and without performing any writes to the database – accounts for almost all the waiting time on disk I/Oas it reads all the data blocks it needs from disk if they are not already in memory. The second phase – the transaction re-execution with strict two-phase locking – accounts for all the waiting time on acquiring locks. The second phase may, though rarely, involve a small waiting time on disk I/Oif a disk block that the transaction needs is flushed to memory (by buffer manager) before the second phase starts.

The technique may increase concurrency as transactions spend almost no time on disk I/Owith locks held and hence locks are held for shorter time. In the first phase the transaction reads all the data items required – and not already in memory – from disk. The locks are acquired in the second phase and the transaction does almost no disk I/Oin this phase. Thus the transaction avoids spending time in disk I/Owith locks held.

The technique may even increase disk throughput as the disk I/Ois not stalled for want of a lock. Consider the following scenario with strict two-phase locking protocol: A transaction is waiting for a lock, the disk is idle and

there are some item to be read from disk. In such a situation disk bandwidth is wasted. But in the proposed technique, the transaction will read all the required item from the disk without acquiring any lock and the disk bandwidth may be properly utilized.

Note that the proposed technique is most useful if the computation involved in the transactions is less and most of the time is spent in disk I/Oand waiting on locks, as is usually the case in disk-resident databases. If the transaction is computation intensive, there may be wasted work. An optimization is to save the updates of transactions in a temporary buffer, and instead of reexecuting the transaction, to compare the data values of items when they are locked with the values used earlier. If the two values are the same for all items, then the buffered updates of the transaction are executed, instead of reexecuting the entire transaction.

16.24 Under what conditions is it less expensive to avoid deadlock than to allow deadlocks to occur and then to detect them?

Answer: Deadlock avoidance is preferable if the consequences of abort are serious (as in interactive transactions), and if there is high contention and a resulting high probability of deadlock.

16.25 If deadlock is avoided by deadlock avoidance schemes, is starvation still possible? Explain your answer.

Answer: A transaction may become the victim of deadlock-prevention rollback arbitrarily many times, thus creating a potential starvation situation.

16.26 Consider the timestamp ordering protocol, and two transactions, one that writes two data items p and q, and another that reads the same two data items. Give a schedule whereby the timestamp test for a write operation fails and causes the first transaction to be restarted, in turn causing a cascading abort of the other transaction. Show how this could result in starvation of both transactions. (Such a situation, where two or more processes carry out actions, but are unable to complete their task because of interaction with the other processes, is called a **livelock**.)

Answer: Consider two transactions T_1 and T_2 shown below.

T1	T2
write(p)	
	read(p)
	read(q)
write(q)	

Let TS(T_1) $<$ TS(T_2) and let the timestamp test at each operation except write(q) be successful. When transaction T_1 does the timestamp test for write(q) it finds that TS(T_1) $<$ R-timestamp(q), since TS(T_1) $<$ TS(T_2) and R-timestamp(q) $=$ TS(T_2). Hence the writeoperation fails and transaction T_1 rolls back. The cascading results in transaction T_2 also being rolled back as it uses the value for item p that is written by transaction T_1.

If this scenario is exactly repeated every time the transactions are restarted, this could result in starvation of both transactions.

16.27 Explain the phantom phenomenon. Why may this phenomenon lead to an incorrect concurrent execution despite the use of the two-phase locking protocol?

Answer: The phantom phenomenon arises when, due to an insertion or deletion, two transactions logically conflict despite not locking any data items in common. The insertion case is described in the book. Deletion can also lead to this phenomenon. Suppose T_i deletes a tuple from a relation while T_j scans the relation. If T_i deletes the tuple and then T_j reads the relation, T_i should be serialized before T_j. Yet there is no tuple that both T_i and T_j conflict on.

An interpretation of 2PL as just locking the accessed tuples in a relation is incorrect. There is also an index or a relation data that has information about the tuples in the relation. This information is read by any transaction that scans the relation, and modified by transactions that update, or insert into, or delete from the relation. Hence locking must also be performed on the index or relation data, and this will avoid the phantom phenomenon.

16.28 Devise a timestamp-based protocol that avoids the phantom phenomenon.

Answer: In the text, we considered two approaches to dealing with the phantom phenomenon by means of locking. The coarser granularity approach obviously works for timestamps as well. The B^+-tree index based approach can be adapted to timestamping by treating index buckets as data items with timestamps associated with them, and requiring that all read accesses use an index. We now show that this simple method works. Suppose a transaction T_i wants to access all tuples with a particular range of search-key values, using a B^+-tree index on that search-key. T_i will need to read all the buckets in that index which have key values in that range. It can be seen that any delete or insert of a tuple with a key-value in the same range will need to write one of the index buckets read by T_i. Thus the logical conflict is converted to a conflict on an index bucket, and the phantom phenomenon is avoided.

16.29 Explain the reason for the use of degree-two consistency. What disadvantages does this approach have?

Answer: The degree-two consistency avoids cascading aborts and offers increased concurrency but the disadvantage is that it does not guarantee serializability and the programmer needs to ensure it.

16.30 Suppose that we use the tree protocol of Section 16.1.5 to manage concurrent access to a B$^+$-tree. Since a split may occur on an insert that affects the root, it appears that an insert operation cannot release any locks until it has completed the entire operation. Under what circumstances is it possible to release a lock earlier?

Answer: Note: The tree-protocol of Section 16.1.5 which is referred to in this question, is different from the multigranularity protocol of Section 16.4 and the B^+-tree concurrency protocol of Section 16.9.

One strategy for early lock releasing is given here. Going down the tree from the root, if the currently visited node's child is not full, release locks held on all nodes except the current node, request an X-lock on the child node, after getting it release the lock on the current node, and then descend to the child. On the other hand, if the child is full, retain all locks held, request an X-lock on the child, and descend to it after getting the lock. On reaching the leaf node, start the insertion procedure. This strategy results in holding locks only on the full index tree nodes from the leaf upwards, until and including the first non-full node.

An optimization to the above strategy is possible. Even if the current node's child is full, we can still release the locks on all nodes but the current one. But after getting the X-lock on the child node, we split it right away. Releasing the lock on the current node and retaining just the lock on the appropriate split child, we descend into it making it the current node. With this optimization, at any time at most two locks are held, of a parent and a child node.

16.31 Give example schedules to show that if any of lookup, insert or delete do not lock the next key value, the phantom phenomenon could go undetected.
Answer: TO BE SOLVED

CHAPTER 17

Recovery System

This chapter covers failure models and a variety of failure recovery techniques. Recovery in a real-life database systems supporting concurrent transactions is rather complicated. To help the student understand concepts better, the chapter presents recovery models in increasing degree of complexity. The chapter starts with a simple model for recovery, ignoring the issue of concurrency. Later, the model is extended to handle concurrent transactions with strict two-phase locking. Towards the end of the chapter, we present an "advanced" recovery algorithm that supports early release of some kinds of locks to improve concurrency, for example in index structures. Finally, we outline the ARIES algorithm variants of which are widely used in practise. ARIES includes the features of the advanced recovery algorithm, along with several optimizations that speed up recovery greatly.

We recommed that at least sections up to and including Log-Based Recovery (Section 17.4) be covered. Shadow paging (Section 17.5) is not used very widely, but is useful for pedagogical reasons, to show alternatives exist to log-based recovery. Recovery with concurrent transactions (Section 17.6) is an interesting section, and should be covered along with Buffer Management (Section 17.7), if possible. Section 17.9, covering the advanced recovery algorithm and ARIES, should be omitted from all except advanced courses. However, it can be used as independent-study material for well-prepared students even in an introductory course.

There are some points worth noting:

- While reading Section 17.2.2 (Stable storage implementation), it should be recalled from the the discussion in section 11.2.1 that a partial disk block write can be detected with a high probability using checksums.
- In section 17.4.3, even though the model assumed is one where transactions execute serially, the recovery procedure says that for *all* transactions T_k in T that have no $<T_k$ **commit**$>$ record in the log, execute **undo**(T_k). More than one such transaction can exist, because of successive transaction failures.

Changes from 3rd edition:
The main changes are (a) we now cover the ARIES recovery algorithm (Section 17.9.6), and and (b) the section on remote backup systems (Section 17.10) has been moved into this chapter from its earlier position in Chapter 20. The latter change is motivated by the increasing need for high availability, which will be reflected in a greatly increased use of remote backup system. Providing high availability can also be considered a job of the recovery system since the same logs can be used for both tasks.

Exercises

17.1 Explain the difference between the three storage types—volatile, nonvolatile, and stable—in terms of I/O cost.
Answer: Volatile storage is storage which fails when there is a power failure. Cache, main memory, and registers are examples of volatile storage. Nonvolatile storage is storage which retains its content despite power failures. An example is magnetic disk. Stable storage is storage which theoretically survives any kind of failure (short of a complete disaster!). This type of storage can only be approximated by replicating data.

In terms of I/O cost, volatile memory is the fastest and non-volatile storage is typically several times slower. Stable storage is slower than non-volatile storage because of the cost of data replication.

17.2 Stable storage cannot be implemented.

a. Explain why it cannot be.
b. Explain how database systems deal with this problem.

Answer:

a. Stable storage cannot really be implemented because all storage devices are made of hardware, and all hardware is vulnerable to mechanical or electronic device failures.
b. Database systems approximate stable storage by writing data to multiple storage devices simultaneously. Even if one of the devices crashes, the data will still be available on a different device. Thus data loss becomes extremely unlikely.

17.3 Compare the deferred- and immediate-modification versions of the log-based recovery scheme in terms of ease of implementation and overhead cost.
Answer:

- The recovery scheme using a log with deferred updates has the following advantages over the recovery scheme with immediate updates:
  a. The scheme is easier and simpler to implement since fewer operations and routines are needed, i.e., no UNDO.
  b. The scheme requires less overhead since no extra I/O operations need to be done until commit time (log records can be kept in memory the entire time).

c. Since the old values of data do not have to be present in the log-records, this scheme requires less log storage space.

- The disadvantages of the deferred modification scheme are :
  a. When a data item needs to accessed, the transaction can no longer directly read the correct page from the database buffer, because a previous write by the same transaction to the same data item may not have been propagated to the database yet. It might have updated a local copy of the data item and deferred the actual database modification. Therefore finding the correct version of a data item becomes more expensive.
  b. This scheme allows less concurrency than the recovery scheme with immediate updates. This is because write-locks are held by transactions till commit time.
  c. For long transaction with many updates, the memory space occupied by log records and local copies of data items may become too high.

17.4 Assume that immediate modification is used in a system. Show, by an example, how an inconsistent database state could result if log records for a transaction are not output to stable storage prior to data updated by the transaction being written to disk.

Answer: Consider a banking scheme and a transaction which transfers \$50 from account A to account B. The transaction has the following steps:

a. **read**(A,a_1)
b. $a_1 := a_1 - 50$
c. **write**(A,a_1)
d. **read**(B,b_1)
e. $b_1 := b_1 + 50$
f. **write**(B,b_1)

Suppose the system crashes after the transaction commits, but before its log records are flushed to stable storage. Further assume that at the time of the crash the update of A in the third step alone had actually been propagated to disk whereas the buffer page containing B was not yet written to disk. When the system comes up it is in an inconsistent state, but recovery is not possible because there are no log records corresponding to this trabsaction in stable storage.

17.5 Explain the purpose of the checkpoint mechanism. How often should checkpoints be performed? How does the frequency of checkpoints affect

- System performance when no failure occurs
- The time it takes to recover from a system crash
- The time it takes to recover from a disk crash

Answer: Checkpointing is done with log-based recovery schemes to reduce the time required for recovery after a crash. If there is no checkpointing, then the entire log must be searched after a crash, and all transactions undone/redone

from the log. If checkpointing had been performed, then most of the log-records prior to the checkpoint can be ignored at the time of recovery.

Another reason to perform checkpoints is to clear log-records from stable storage as it gets full.

Since checkpoints cause some loss in performance while they are being taken, their frequency should be reduced if fast recovery is not critical. If we need fast recovery checkpointing frequency should be increased. If the amount of stable storage available is less, frequent checkpointing is unavoidable. Checkpoints have no effect on recovery from a disk crash; archival dumps are the equivalent of checkpoints for recovery from disk crashes.

17.6 When the system recovers from a crash (see Section 17.6.4), it constructs an undo-list and a redo-list. Explain why log records for transactions on the undo-list must be processed in reverse order, while those log records for transactions on the redo-list are processed in a forward direction.

Answer: The first phase of recovery is to undo the changes done by the failed transactions, so that all data items which have been modified by them get back the values they had before the *first* of the failed transactions started. If several of the failed transactions had modified the same data item, forward processing of log-records for undo-list transactions would make the data item get the value which it had before the *last* failed transaction to modify that data item started. This is clearly wrong, and we can see that reverse prcessing gets us the desired result.

The second phase of recovery is to redo the changes done by committed transactons, so that all data items which have been modified by them are restored to the value they had after the *last* of the committed transactions finished. It can be seen that only forward processing of log-records belonging to redo-list transactions can guarantee this.

17.7 Compare the shadow-paging recovery scheme with the log-based recovery schemes in terms of ease of implementation and overhead cost.

Answer: The shadow-paging scheme is easy to implement for single-transaction systems, but difficult for multiple-transaction systems. In particular it is very hard to allow multiple updates concurrently on the same page. Shadow paging could suffer from extra space overhead, but garbage collection can take care of that. The I/O overhead for shadow paging is typically higher than the log based schemes, since the log based schemes need to write one record per update to the log, whereas the shadow paging scheme needs to write one block per updated block.

17.8 Consider a database consisting of 10 consecutive disk blocks (block 1, block $2, \ldots,$ block 10). Show the buffer state and a possible physical ordering of the blocks after the following updates, assuming that shadow paging is used, that the buffer in main memory can hold only three blocks, and that a least recently used (LRU) strategy is used for buffer management.

read block 3
read block 7
read block 5
read block 3
read block 1
modify block 1
read block 10
modify block 5

Answer: The initial ordering of the disk blocks is: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Assume that the two blocks following block 10 on the disk, are the first two blocks in the list of free blocks.

a. The first 3 **read** steps result in blocks 3, 7, 5 being placed in the buffer.
b. The 4th **read** step requires no disk access.
c. The 5th **read** step requires block 1 to be read. Block 7 is the least recently used block in the buffer, so it is replaced by block 1.
d. The 6th step is to modify block 1. The first free block is removed from the free block list, and the entry 1 in the *current page table* is made to point to it. Block 1 in the buffer is modified. When dirty blocks are flushed back to disk at the time of transaction commit, they should be written to the disk blocks pointed to the updated current page table.
e. The 7th step causes block 10 to be read. Block 5 is overwritten in the buffer since it is the least recently used.
f. In the 8th step, block 3 is replaced by block 5, and then block 5 is modified as in the 6th step.

Therefore the final disk ordering of blocks is: 2, 3, 4, 6, 7, 8, 9, 10, 1, 5. The set of blocks in the buffer are: 5 (modified), 10, 1 (modified). These must be flushed to the respective disk blocks as pointed to by the current page table, before the transaction performs commit processing.

17.9 Explain how the buffer manager may cause the database to become inconsistent if some log records pertaining to a block are not output to stable storage before the block is output to disk.
Answer: If a data item x is modified on disk by a transaction before the corresponding log record is written to stable storage, then the only record of the old value of x is in main memory where it would be lost in a crash. If the transaction had not yet finished at the time of the crash, an unrecoverable inconsistency will result.

17.10 Explain the benefits of logical logging. Give examples of one situation where logical logging is preferable to physical logging and one situation where physical logging is preferable to logical logging.
Answer: Logical logging has less log space requirement, and with logical undo logging it allows early release of locks. This is desirable in situations like concurrency control for index structures, where a very high degree of concurrency is required. An advantage of employing physical redo logging is that fuzzy

checkpoints are possible. Thus in a system which needs to perform frequent checkpoints, this reduces checkpointing overhead.

17.11 Explain the reasons why recovery of interactive transactions is more difficult to deal with than is recovery of batch transactions. Is there a simple way to deal with this difficulty? (Hint: Consider an automatic teller machine transaction in which cash is withdrawn.)

Answer: Interactive transactions are more difficult to recover from than batch transactions because some actions may be irrevocable. For example, an output (write) statement may have fired a missile, or caused a bank machine to give money to a customer. The best way to deal with this is to try to do all output statements at the end of the transaction. That way if the transaction aborts in the middle, no harm will be have been done.

17.12 Sometimes a transaction has to be undone after it has commited, because it was erroneously executed, for example because of erroneous input by a bank teller.

a. Give an example to show that using the normal transaction undo mechanism to undo such a transaction could lead to an inconsistent state.

b. One way to handle this situation is to bring the whole database to a state prior to the commit of the erroneous transaction (called *point-in-time* recovery). Transactions that committed later have their effects rolled back with this scheme.

Suggest a modification to the advanced recovery mechanism to implement point-in-time recovery.

c. Later non-erroneous transactions can be reexecuted logically, but cannot be reexecuted using their log records. Why?

Answer:

- Consider the a bank account A with balance \$100. Consider two transactions T_1 and T_2 each depositing \$10 in the account. Thus the balance would be \$120 after both these transactions are executed. Let the transactions execute in sequence: T_1 first and then T_2. The log records corresponding to the updates of A by transactions T_1 and T_2 would be $< T_1, A, 100, 110 >$ and $< T_2, A, 110, 120 >$ resp.

  Say, we wish to undo transaction T_1. The normal transaction undo mechanism will replaces the value in question – A in this example – by the old-value field in the log record. Thus if we undo transaction T_1 using the normal transaction undo mechanism the resulting balance would be \$100 and we would, in effect, undo both transactions, whereas we intend to undo only transaction T_1.
- Let the erroneous transaction be T_e.
  - ☐ Identify the latest checkpoint, say C, in the log before the log record $< T_e, START >$.
  - ☐ Redo all log records starting from the checkpoint C till the log record $< T_e, COMMIT >$. Some transaction – apart from transaction T_e –

would be active at the commit time of transaction T_e. Let S_1 be the set of such transactions.

- ☐ Rollback T_e and the transactions in the set S_1.
- ☐ Scan the log further starting from the log record $< T_e, COMMIT >$ till the end of the log. Note the transactions that were started after the commit point of T_e. Let the set of such transactions be S_2. Re-execute the transactions in set S_1 and S_2 logically.

- Consider again an example from the first item. Let us assume that both transactions are undone and the balance is reverted back to the original value \$100.

  Now we wish to redo transaction T_2. If we redo the log record $< T_2, A, 110, 120 >$ corresponding to transaction T_2 the balance would become \$120 and we would, in effect, redo both transactions, whereas we intend to redo only transaction T_2.

17.13 Logging of updates is not done explicitly in persistent programming languages. Describe how page access protections provided by modern operating systems can be used to create before and after images of pages that are updated. (Hint: See Exercise 16.12.)

Answer: This is implemented by using `mprotect` to initially turn off access to all pages, for the process. When the process tries to access an address in a page, a protection violation occurs. The handler accociated with protection violation then requests a write lock on the page, and after the lock is acquired, it writes the initial contents (before-image) of the page to the log. It then uses `mprotect` to allow write access to the page by the process, and finally allows the process to continue. When the transaction is ready to commit, and before it releases the lock on the page, it writes the contents of the page (after-image) to the log. These before- and after- images can be used for recovery after a crash.

This scheme can be optimized to not write the whole page to log for undo logging, provided the program pins the page in memory.

17.14 ARIES assumes there is space in each page for an LSN. When dealing with large objects that span multiple pages, such as operating system files, an entire page may be used by an object, leaving no space for the LSN. Suggest a technique to handle such a situation; your technique must support physical redos but need not support physiological redos.

Answer: We can maintain the LSNs of such pages in an array in a separate disk page. The LSN entry of a page on the disk is the sequence number of the latest log record reflected on the disk. In the normal case, as the LSN of a page resides in the page itself, the page and its LSN are in consistent state. But in the modified scheme as the LSN of a page resides in a separate page it may not be written to the disk at a time when the actual page is written and thus the two may not be in consistent state.

If a page is written to the disk before its LSN is updated on the disk and the system crashes then, during recovery, the page LSN read from the LSN array from the disk is older than the sequence number of the log record reflected to the disk. Thus some updates on the page will be redone unnecessarily but

this is fine as updates are idempotent. But if the page LSN is written to the disk to before the actual page is written and the system crashes then some of the updates to the page may be lost. The sequence number of the log record corresponding to the latest update to the page that made to the disk is older than the page LSN in the LSN array and all updates to the page between the two LSNs are lost.

Thus the LSN of a page should be written to the disk only after the page has been written and; we can ensure this as follows: before writing a page containing the LSN array to the disk, we should flush the corresponding pages to the disk. (We can maintain the page LSN at the time of the last flush of each page in the buffer separately, and avoid flushing pages that have been flushed already.)

17.15 Explain the difference between a system crash and a "disaster."

Answer: In a system crash, the CPU goes down, and disk may also crash. But stable-storage at the site is assumed to survive system crashes. In a "disaster", *everything* at a site is destroyed. Stable storage needs to be distributed to survive disasters.

17.16 For each of the following requirements, identify the best choice of degree of durability in a remote backup system:

a. Data loss must be avoided but some loss of availability may be tolerated.
b. Transaction commit must be accomplished quickly, even at the cost of loss of some committed transactions in a disaster.
c. A high degree of availability and durability is required, but a longer running time for the transaction commit protocol is acceptable.

Answer:

a. Two very safe is suitable here because it guarantees durability of updates by committed transactions, though it can proceed only if both primary and backup sites are up. Availability is low, but it is mentioned that this is acceptable.
b. One safe committing is fast as it does not have to wait for the logs to reach the backup site. Since data loss can be tolerated, this is the best option.
c. With two safe committing, the probability of data loss is quite low, and also commits can proceed as long as at least the primary site is up. Thus availability is high. Commits take more time than in the one safe protocol, but that is mentioned as acceptable.

CHAPTER 18

Database System Architectures

The chapter is suitable for an introductory course. We recommend covering it, at least as self-study material, since students are quite likely to use the non-centralized (particularly client-server) database architectures when they enter the real world. The material in this chapter could potentially be supplemented by the two-phase commit protocol (2PC), (Section 19.4.1 from Chapter 19) to give students an overview of the most important details of non-centralized database architectures.

Changes from 3rd edition:
Coverage of database process structures (Section 18.2.1) is new in this edition. Coverage of network technology has been updated, and storage area networks are briefly covered.

Exercises

18.1 Why is it relatively easy to port a database from a single processor machine to a multiprocessor machine if individual queries need not be parallelized?
Answer: Porting is relatively easy to a shared memory multiprocessor machine. Databases designed for single-processor machines already provide multitasking, allowing multiple processes to run on the same processor in a time-shared manner, giving a view to the user of multiple processes running in parallel. Thus, coarse-granularity parallel machines logically appear to be identical to single-processor machines, making the porting relatively easy.

Porting a database to a shared disk or shared nothing multiprocessor architecture is a little harder.

18.2 Transaction server architectures are popular for client-server relational databases, where transactions are short. On the other hand, data server architectures are popular for client-server object-oriented database systems, where transactions are expected to be relatively long. Give two reasons why data servers

may be popular for object-oriented databases but not for relational databases.
Answer: Data servers are good if data transfer is small with respect to computation, which is often the case in applications of OODBs such as computer aided design. In contrast, in typical relational database applications such as transaction processing, a transaction performs little computation but may touch several pages, which will result in a lot of data transfer with little benefit in a data server architecture. Another reason is that structures such as indices are heavily used in relational databases, and will become spots of contention in a data server architecture, requiring frequent data transfer. There are no such points of frequent contention in typical current-day OODB applications such as computer aided design.

18.3 Instead of storing shared structures in shared memory, an alternative architecture would be to store them in the local memory of a special process, and access the shared data by interprocess communication with the process. What would be the drawback of such an architecture?
Answer: The drawbacks would be that two interprocess messages would be required to acquire locks, one for the request and one to confirm grant. Interprocess communication is much more expensive than memory access, so the cost of locking would increase. The process storing the shared structures could also become a bottleneck.

The benefit of this alternative is that the lock table is protected better from erroneous updates since only one process can access it.

18.4 In typical client–server systems the server machine is much more powerful than the clients; that is, its processor is faster, it may have multiple processors, and it has more memory and disk capacity. Consider instead a scenario where client and server machines have exactly the same power. Would it make sense to build a client–server system in such a scenario? Why? Which scenario would be better suited to a data-server architecture?
Answer: With powerful clients, it still makes sense to have a client-server system, rather than a fully centralized system. If the data-server architecture is used, the powerful clients can off-load all the long and compute intensive transaction processing work from the server, freeing it to perform only the work of satisfying read-write requests. even if the transaction-server model is used, the clients still take care of the user-interface work, which is typically very compute-intensive.

A fully distributed system might seem attractive in the presence of powerful clients, but client-server systems still have the advantage of simpler concurrency control and recovery schemes to be implemented on the server alone, instead of having these actions distributed in all the machines.

18.5 Consider an object-oriented database system based on a client-server architecture, with the server acting as a data server.

a. What is the effect of the speed of the interconnection between the client and the server on the choice between object and page shipping?

b. If page shipping is used, the cache of data at the client can be organized either as an object cache or a page cache. The page cache stores data in units of a page, while the object cache stores data in units of objects. Assume objects are smaller than a page. Describe one benefit of an object cache over a page cache.

Answer:

a. We assume that objects are smaller than a page and fit in a page. If the interconnection link is slow it is better to choose object shipping, as in page shipping a lot of time will be wasted in shipping objects that might never be needed. With a fast interconnection though, the communication overheads and latencies, not the actual volume of data to be shipped, becomes the bottle neck. In this scenario page shipping would be preferable.

b. Two benefits of an having an object-cache rather than a page-cache, even if page shipping is used, are:-

i. When a client runs out of cache space, it can replace objects without replacing entire pages. The reduced caching granularity might result in better cache-hit ratios.

ii. It is possible for the server to ask clients to return some of the locks which they hold, but don't need (lock de-escalation). Thus there is scope for greater concurrency. If page caching is used, this is not possible.

18.6 What is lock de-escalation, and under what conditions is it required? Why is it not required if the unit of data shipping is an item?

Answer: In a client-server system with page shipping, when a client requests an item, the server typically grants a lock not on the requested item, but on the *page* having the item, thus implicitly granting locks on all the items in the page. The other items in the page are said to be *prefetched*. If some other client subsequently requests one of the prefetched items, the server may ask the owner of the page lock to transfer back the lock on this item. If the page lock owner doesn't need this item, it de-escalates the page lock that it holds, to item locks on all the items that it is actually accessing, and then returns the locks on the unwanted items. The server can then grant the latter lock request.

If the unit of data shipping is an item, there are no coarser granularity locks; even if prefetching is used, it is typically implemented by granting individual locks on each of the prefetched items. Thus when the server asks for a return of a lock, there is no question of de-escalation, the requested lock is just returned if the client has no use for it.

18.7 Suppose you were in charge of the database operations of a company whose main job is to process transactions. Suppose the company is growing rapidly each year, and has outgrown its current computer system. When you are choosing a new parallel computer, what measure is most relevant—speedup, batch scaleup, or transaction scaleup? Why?

Answer: With increasing scale of operations, we expect that the number of

transactions submitted per unit time increases. On the other hand, we wouldn't expect most of the individual transactions to grow longer, nor would we require that a given transaction should execute more quickly now than it did before. Hence transaction scale-up is the most relevant measure in this scenario.

18.8 Suppose a transaction is written in C with embedded SQL, and about 80 percent of the time is spent in the SQL code, with the remaining 20 percent spent in C code. How much speedup can one hope to attain if parallelism is used only for the SQL code? Explain.
Answer: Since the part which cannot be parallelized takes 20% of the total running time, the best speedup we can hope for has to be less than 5.

18.9 What are the factors that can work against linear scaleup in a transaction processing system? Which of the factors are likely to be the most important in each of the following architectures: shared memory, shared disk, and shared nothing?
Answer: Increasing contention for shared resources prevents linear scale-up with increasing parallelism. In a shared memory system, contention for memory (which implies bus contention) will result in falling scale-up with increasing parallelism. In a shared disk system, it is contention for disk and bus access which affects scale-up. In a shared-nothing system, inter-process communication overheads will be the main impeding factor. Since there is no shared memory, acquiring locks, and other activities requiring message passing between processes will take more time with increased parallelism.

18.10 Consider a bank that has a collection of sites, each running a database system. Suppose the only way the databases interact is by electronic transfer of money between one another. Would such a system qualify as a distributed database? Why?
Answer: In a distributed system, all the sites typically run the same database management software, and they share a global schema. Each site provides an environment for execution of both global transactions initiated at remote sites and local transactions. The system described in the question does not have these properties, and hence it cannot qualify as a distributed database.

18.11 Consider a network based on dial-up phone lines, where sites communicate periodically, such as every night. Such networks are often configured with a server site and multiple client sites. The client sites connect only to the server, and exchange data with other clients by storing data at the server and retrieving data stored at the server by other clients. What is the advantage of such an architecture over one where a site can exchange data with another site only by first dialing it up?
Answer: With the central server, each site does not have to remember which site to contact when a particular data item is to be requested. The central server alone needs to remember this, so data items can be moved around easily, depending on which sites access which items most frequently. Other house-keeping tasks are also centralized rather than distributed, making the system easier to develop and maintain. Of course there is the disadvantage of a total shutdown

in case the server becomes unavailable. Even if it is running, it may become a bottleneck because every request has to be routed via it.

CHAPTER 19

Distributed Databases

Distributed databases in general, and *heterogeneous* distributed databases in particular, are of increasing practical importance, as organizations attempt to integrate databases across physical and organizational boundaries. Such interconnection of databases to create a distributed or multidatabase is in fact proving crucial to competitiveness for many companies. This chapter reconsiders the issues addressed earlier in the text, such as query processing, recovery and concurrency control, from the standpoint of distributed databases.

This is a long chapter, and is appropriate only for an advanced course. Single topics may be chosen for inclusion in an introductory course. Good choices include distributed data storage, heterogeneity and two-phase commit.

Changes from 3rd edition:
This chapter has changed significantly from the previous edition.

- The emphasis on transparency in the earlier edition has been dropped, and instead the chapter begins by considering the distinction between heterogeneous and homogeneous distributed databases.
- All details of three phase commit have been dropped since it is not widely used in practise.
- We have introduced coverage of alternative models of transaction processing in Section 19.4.3, with emphasis on the persistent messaging based approach to distributed transactions.
- Replication with weak levels of consistency, which is widely used in practise, is now covered in Section 19.5.3.
- Distributed algorithms for deadlock detection has been dropped since they are too complicated and expensive to be practical.

- We have introduced detailed coverage of failure handing for providing high availability in distributed databases (Section 19.6).
- Heterogeneous databases are now covered in more detail in Section 19.8, while details of weak levels of serializability in multidatabases have been moved to Chapter 24.
- Coverage of directory systems, with emphasis on LDAP, has been introduced in this edition (Section 19.9).

Exercises

19.1 Discuss the relative advantages of centralized and distributed databases.
Answer:

- A distributed database allows a user convenient and transparent access to data which is not stored at the site, while allowing each site control over its own local data. A distributed database can be made more reliable than a centralized system because if one site fails, the database can continue functioning, but if the centralized system fails, the database can no longer continue with its normal operation. Also, a distributed database allows parallel execution of queries and possibly splitting one query into many parts to increase throughput.
- A centralized system is easier to design and implement. A centralized system is cheaper to operate because messages do not have to be sent.

19.2 Explain how the following differ: fragmentation transparency, replication transparency, and location transparency.
Answer:

a. With fragmentation transparency, the user of the system is unaware of any fragmentation the system has implemented. A user may formulate queries against global relations and the system will perform the necessary transformation to generate correct output.
b. With replication transparency, the user is unaware of any replicated data. The system must prevent inconsistent operations on the data. This requires more complex concurrency control algorithms.
c. Location transparency means the user is unaware of where data are stored. The system must route data requests to the appropriate sites.

19.3 How might a distributed database designed for a local-area network differ from one designed for a wide-area network?
Answer: Data transfer on a local-area network (LAN) is much faster than on a wide-area network (WAN). Thus replication and fragmentation will not increase throughput and speed-up on a LAN, as much as in a WAN. But even in a LAN, replication has its uses in increasing reliability and availability.

19.4 When is it useful to have replication or fragmentation of data? Explain your answer.
Answer: Replication is useful when there are many read-only transactions at

different sites wanting access to the same data. They can all execute quickly in parallel, accessing local data. But updates become difficult with replication. Fragmentation is useful if transactions on different sites tend to access different parts of the database.

19.5 Explain the notions of transparency and autonomy. Why are these notions desirable from a human-factors standpoint?

Answer: Autonomy is the amount of control a single site has over the local database. It is important because users at that site want quick and correct access to local data items. This is especially true when one considers that local data will be most frequently accessed in a database. Transparency hides the distributed nature of the database. This is important because users should not be required to know about location, replication, fragmentation or other implementation aspects of the database.

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

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

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

Answer:

a. The types of failure that can occur in a distributed system include
   i. Computer failure (site failure).
   ii. Disk failure.
   iii. Communication failure.

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

19.7 Consider a failure that occurs during 2PC for a transaction. For each possible failure that you listed in Exercise 19.6a, explain how 2PC ensures transaction atomicity despite the failure.

Answer: A proof that 2PC guarantees atomic commits/aborts inspite of site and link failures, follows. The main idea is that after all sites reply with a <**ready** T> message, only the co-ordinator of a transaction can make a commit or abort decision. Any subsequent commit or abort by a site can happen only after it ascertains the co-ordinator's decision, either directly from the co-ordinator, or indirectly from some other site. Let us enumerate the cases for a site aborting, and then for a site committing.

a. A site can abort a transaction T (by writing an <**abort** T> log record) only under the following circumstances:-
   i. It has not yet written a <**ready** T> log-record. In this case, the co-ordinator could not have got, and will not get a <**ready** T> or <**commit** T> message from this site. Therefore only an abort decision can be made by the co-ordinator.
   ii. It has written the <**ready** T> log record, but on inquiry it found out that some other site has an <**abort** T> log record. In this case it is

correct for it to abort, because that other site would have ascertained the co-ordinator's decision (either directly or indirectly) before actually aborting.

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

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

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

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

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

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

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

Answer: Site A cannot distinguish between the three cases until communication has resumed with site B. The action which it performs while B is inaccessible must be correct irrespective of which of these situations has actually occurred, and must be such that B can re-integrate consistently into the distributed system once communication is restored.

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

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

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

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

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

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

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

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

19.11 If we apply a distributed version of the multiple-granularity protocol of Chapter 16 to a distributed database, the site responsible for the root of the DAG may become a bottleneck. Suppose we modify that protocol as follows:

- Only intention-mode locks are allowed on the root.
- All transactions are given all possible intention-mode locks on the root automatically.

Show that these modifications alleviate this problem without allowing any nonserializable schedules.

Answer: Serializability is assured since we have not changed the rules for the multiple granularity protocol. Since transactions are automatically granted all intention locks on the root node, and are not given other kinds of locks on it, there is no need to send any lock requests to the root. Thus the bottleneck is relieved.

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

Answer: In remote backup systems all transactions are performed at the primary site and the data is replicated at the remote backup site. The remote backup site is kept synchronized with the updates at the primary site by sending all log records. Whenever the primary site fails, the remote backup site takes over processing.

The distributed systems offer greater availability by having multiple copies of the data at different sites whereas the remote backup systems offer lesser availability at lower cost and execution overhead.

In a distributed system, transaction code runs at all the sites whereas in a remote backup system it runs only at the primary site. The distributed system transactions follow two-phase commit to have the data in consistent state whereas a remote backup system does not follow two-phase commit and avoids related overhead.

19.13 Give an example where lazy replication can lead to an inconsistent database state even when updates get an exclusive lock on the primary (master) copy.
Answer: Consider the balance in an account, replicated at N sites. Let the current balance be $\$100$ – consistent across all sites. Consider two transactions T_1 and T_2 each depositing $\$10$ in the account. Thus the balance would be $\$120$ after both these transactions are executed. Let the transactions execute in sequence: T_1 first and then T_2. Suppose the copy of the balance at one of the sites, say s, is not consistent – due to lazy replication strategy – with the primary copy after transaction T_1 is executed and let transaction T_2 read this copy of the balance. One can see that the balance at the primary site would be $\$110$ at the end.

19.14 Study and summarize the facilities that the database system you are using provides for dealing with inconsistent states that can be reached with lazy propagation of updates.
Answer: TO BE FILLED IN.

19.15 Discuss the advantages and disadvantages of the two methods that we presented in Section 19.5.2 for generating globally unique timestamps.
Answer: The centralized approach has the problem of a possible bottleneck at the central site and the problem of electing a new central site if it goes down. The distributed approach has the problem that many messages must be exchanges to keep the system fair, or one site can get ahead of all other sites and dominate the database.

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

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

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

- The graph contains a vertex for every transaction in the system.

- The graph has an edge (T_i, T_j) if and only if
  - There is an edge (T_i, T_j) in one of the wait-for graphs.
  - An edge (T_i, T_j, n) (for some n) appears in more than one wait-for graph.

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

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

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

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

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

19.17 Consider a relation that is fragmented horizontally by *plant-number*:

employee (*name, address, salary, plant-number*)

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

a. Find all employees at the Boca plant.
b. Find the average salary of all employees.
c. Find the highest-paid employee at each of the following sites: Toronto, Edmonton, Vancouver, Montreal.
d. Find the lowest-paid employee in the company.

Answer:

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

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

19.18 Consider the relations

employee (*name, address, salary, plant-number*)
machine (*machine-number, type, plant-number*)

Assume that the *employee* relation is fragmented horizontally by *plant-number*, and that each fragment is stored locally at its corresponding plant site. Assume that the *machine* relation is stored in its entirety at the Armonk site. Describe a good strategy for processing each of the following queries.

a. Find all employees at the plant that contains machine number 1130.
b. Find all employees at plants that contain machines whose type is "milling machine."
c. Find all machines at the Almaden plant.
d. Find employee ⋈ machine.

Answer:

a. i. Perform $\Pi_{plant\text{-}number}\ (\sigma_{machine\text{-}number=1130}\ (machine))$ at Armonk.
ii. Send the query $\Pi_{name}\ (employee)$ to all site(s) which are in the result of the previous query.
iii. Those sites compute the answers.
iv. Union the answers at the destination site.
b. This strategy is the same as 0.a, except the first step should be to perform
$\Pi_{plant\text{-}number}\ (\sigma_{type=\text{“milling machine”}}\ (machine))$ at Armonk.
c. i. Perform $\sigma_{plant\text{-}number\ =\ x}\ (machine)$ at Armonk, where x is the plant-number for Almaden.
ii. Send the answers to the destination site.
d. Strategy 1:
i. Group *machine* at Armonk by plant number.
ii. Send the groups to the sites with the corresponding plant-number.
iii. Perform a local join between the local data and the received data.
iv. Union the results at the destination site.
Strategy 2:
Send the *machine* relation at Armonk, and all the fragments of the *employee* relation to the destination site. Then perform the join at the destination site.

There is parallelism in the join computation according to the first strategy but not in the second. Nevertheless, in a WAN the amount of data to be shipped is the main cost factor. We expect that each plant will have more than one machine, hence the result of the local join at each site will be a cross-product of the employee tuples and machines at that plant. This cross-product's size is greater than the size of the *emplyee* fragment at that site. As a result the second strategy will result in less data shipping, and will be more efficient.

19.19 For each of the strategies of Exercise 19.18, state how your choice of a strategy depends on:

a. The site at which the query was entered
b. The site at which the result is desired

Answer:

a. Assuming that the cost of shipping the query itself is minimal, the site at which the query was submitted does not affect our strategy for query evaluation.
b. For the first query, we find out the plant numbers where the machine number 1130 is present, at Armonk. Then the employee tuples at all those plants are shipped to the destination site. We can see that this strategy is more or less independent of the destination site. The same can be said of the second query. For the third query, the selection is performed at Armonk and results shipped to the destination site. This stratgy is obviously independent of the destination site.

For the fourth query, we have two strategies. The first one performs local joins at all the plant sites and their results are unioned at the destination site. In the second strategy, the *machine* relation at Armonk as well as all the fragments of the *employee* relation are first shipped to the destination, where the join operation is performed. There is no obvious way to optimize these two strategies based on the destination site. In the answer to Exercise 19.18 we saw the reason why the second strategy is expected to result in less data shipping than the first. That reason is independent of destination site, and hence we can in general prefer strategy two to strategy one, regardless of the destination site.

19.20 Compute $r \ltimes s$ for the relations of Figure 19.1.

Answer: The result is as follows.

$r \ltimes s =$

A	B	C
1	2	3
5	3	2

19.21 Is $r_i \ltimes r_j$ necessarily equal to $r_j \ltimes r_i$? Under what conditions does $r_i \ltimes r_j = r_j \ltimes r_i$ hold?
Answer: In general, $r_i \ltimes r_j \neq r_j \ltimes r_i$. This can be easily seen from

A	B	C
1	2	3
4	5	6
1	2	4
5	3	2
8	9	7

r

C	D	E
3	4	5
3	6	8
2	3	2
1	4	1
1	2	3

s

Figure 19.1 Relations for Exercise 19.20.

Exercise 19.20, in which $r \ltimes s \neq s \ltimes r$. $r \ltimes s$ was given in 19.20, while

$s \ltimes r =$

C	D	E
3	4	5
3	6	8
2	3	2

By definition, $r_i \ltimes r_j = \Pi_{R_i}(r_i \bowtie r_j)$ and $r_j \ltimes r_i = \Pi_{R_j}(r_i \bowtie r_j)$, where R_i and R_j are the schemas of r_i and r_j respectively. For $\Pi_{R_i}(r_i \bowtie r_j)$ to be always equal to $\Pi_{R_j}(r_i \bowtie r_j)$, the schemas R_i and R_j must be the same.

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

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

19.23 Describe how LDAP can be used to provide multiple hierarchical views of data, without replicating the base level data.
Answer: This can be done using referrals. For example an organization may maintain its information about departments either by geography (i.e. all departments in a site of the the organization) or by structure (i.e. information about a department from all sites). These two hierarchies can be maintained by defining two different schemas with department information at a site as the base information. The entries in the two hierarchies will refer to the base information entry using referrals.

CHAPTER 20

Parallel Databases

This chapter is suitable for an advanced course, but can also be used for independent study projects by students of a first course. The chapter covers several aspects of the design of parallel database systems — partitioning of data, parallelization of individual relational operations, and parallelization of relational expressions. The chapter also briefly covers some systems issues, such as cache coherency and failure resiliency.

The most important applications of parallel databases today are for warehousing and analyzing large amounts of data. Therefore partitioning of data and parallel query processing are covered in significant detail. Query optimization is also of importance, for the same reason. However, parallel query optimization is still not a fully solved problem; exhaustive search, as is used for sequential query optimization, is too expensive in a parallel system, forcing the use of heuristics. Thus parallel query optimization is an area of ongoing research.

The description of parallel query processing algorithms is based on the shared-nothing model. Students may be asked to study how the algorithms can be improved if shared-memory machines are used instead.

Changes from 3rd edition:
There are no major changes from the previous edition.

Exercises

20.1 For each of the three partitioning techniques, namely round-robin, hash partitioning, and range partitioning, give an example of a query for which that partitioning technique would provide the fastest response.

Answer:

Round robin partitioning:

When relations are large and queries read entire relations, round-robin gives good speed-up and fast response time.

Hash partitioning
: For point queries, this gives the fastest response, as each disk can process a query simultaneously. If the hash partitioning is uniform, even entire relation scans can be performed efficiently.

Range partitioning
: For range queries which access a few tuples, this gives fast response.

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

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

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

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

20.3 What factors could result in skew when a relation is partitioned on one of its attributes by:

a. Hash partitioning
b. Range partitioning

In each case, what can be done to reduce the skew?

Answer:

a. Hash-partitioning:
: Too many records with the same value for the hashing attribute, or a poorly chosen hash function without the properties of randomness and uniformity, can result in a skewed partition. To improve the situation, we should experiment with better hashing functions for that relation.

b. Range-partitioning:
: Non-uniform distribution of values for the partitioning attribute (including duplicate values for the partitioning attribute) which are not taken into account by a bad partitioning vector is the main reason for skewed partitions. Sorting the relation on the partitioning attribute and then dividing it into n ranges with equal number of tuples per range will give a good partitioning vector with very low skew.

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

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

Answer:

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

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

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

a. Explain why.

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

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

Answer:

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

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

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

20.6 Give an example of a join that is not a simple equi-join for which partitioned parallelism can be used. What attributes should be used for partitioning?
Answer: We give two examples of such joins.

a. $r \bowtie_{(r.A=s.B) \wedge (r.A<s.C)} s$
Here we have extra conditions which can be checked after the join. Hence partitioned parallelism is useful.

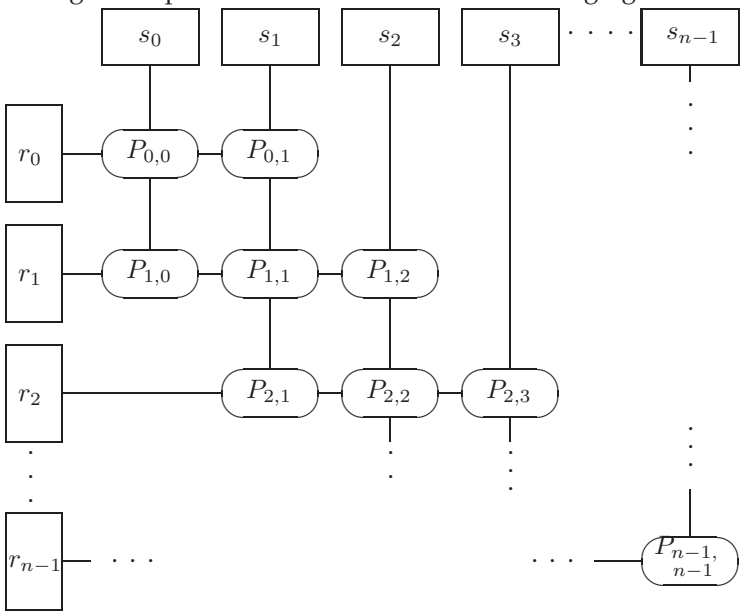
b. $r \bowtie_{(r.A \geq (\lfloor s.B/20 \rfloor)*20) \wedge (r.A<((\lfloor s.B/20 \rfloor)+1)*20)} s$
This is a query in which an r tuple and an s tuple join with each other if they fall into the same range of values. Hence partitioned parallelism applies naturally to this scenario.

For both the queries, r should be partitioned on attribute A and s on attribute B.

20.7 Consider join processing using symmetric fragment and replicate with range partitioning. How can you optimize the evaluation if the join condition is of

the form $| r.A - s.B | \leq k$, where k is a small constant. Here, $| x |$ denotes the absolute value of x. A join with such a join condition is called a **band join**.

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



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

20.8 Describe a good way to parallelize each of the following.

- **a.** The difference operation
- **b.** Aggregation by the **count** operation
- **c.** Aggregation by the **count distinct** operation
- **d.** Aggregation by the **avg** operation
- **e.** Left outer join, if the join condition involves only equality
- **f.** Left outer join, if the join condition involves comparisons other than equality
- **g.** Full outer join, if the join condition involves comparisons other than equality

Answer:

- **a.** We can parallelize the difference operation by partitioning the relations on all the attributes, and then computing differences locally at each processor. As in aggregation, the cost of transferring tuples during partitioning can

be reduced by partially computing differences at each processor, before partitioning.

b. Let us refer to the group-by attribute as attribute A, and the attribute on which the aggregation function operates, as attribute B. **count** is performed just like **sum** (mentioned in the book) except that, a count of the number of values of attribute B for each value of attribute A) is transferred to the correct destination processor, instead of a sum. After partitioning, the partial counts from all the processors are added up locally at each processor to get the final result.

c. For this, partial counts cannot be computed locally before partitioning. Each processor instead transfers all unique B values for each A value to the correct destination processor. After partitioning, each processor locally counts the number of unique tuples for each value of A, and then outputs the final result.

d. This can again be implemented like **sum**, except that for each value of A, a **sum** of the B values as well as a **count** of the number of tuples in the group, is transferred during partitioning. Then each processor outputs its local result, by dividing the total sum by total number of tuples for each A value assigned to its partition.

e. This can be performed just like partitioned natural join. After partitioning, each processor computes the left outer join locally using any of the strategies of Chapter 13.

f. The left outer join can be computed using an extension of the Fragment-and-Replicate scheme to compute non equi-joins. Consider $r \mathbin{⟕} s$. The relations are partitioned, and $r \bowtie s$ is computed at each site. We also collect tuples from r that did not match any tuples from s; call the set of these dangling tuples at site i as d_i. After the above step is done at each site, for each fragment of r, we take the intersection of the d_i's from every processor in which the fragment of r was replicated. The intersections give the real set of dangling tuples; these tuples are padded with nulls and added to the result. The intersections themselves, followed by addition of padded tuples to the result, can be done in parallel by partitioning.

g. The algorithm is basically the same as above, except that when combining results, the processing of dangling tuples must done for both relations.

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

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

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

Answer:

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

b. Let the histogram ranges be called $h_1, h_2, \ldots, h_h$, and the partitions $p_1, p_2, \ldots, p_p$. Let the frequencies of the histogram ranges be $n_1, n_2, \ldots, n_h$. Each partition should contain N/p tuples, where $N = \Sigma_{i=1}^{h} n_i$.

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

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

where

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

20.10 Describe the benefits and drawbacks of pipelined parallelism.

Answer:

Benefits:

No need to write intermediate relations to disk only to read them back immediately.

Drawbacks:

a. Cannot take advantage of high degrees of parallelism, as typical queries do not have large number of operations.

b. Not possible to pipeline operators which need to look at all the input before producing any output.

c. Since each operation executes on a single processor, the most expensive ones take a long time to finish. Thus speed-up will be low inspite of parallelism.

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

a. Why is it a good idea to partition the copies of the data items of a processor across multiple processors?

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

Answer:

a. The copies of the data items at a processor should be partitioned across multiple other processors, rather than stored in a single processor, for the following reasons:
- to better distribute the work which should have been done by the failed processor, among the remaining processors.
- Even when there is no failure, this technique can to some extent deal with hot-spots created by read only transactions.

b. RAID level 0 itself stores an extra copy of each data item (mirroring). Thus this is similar to mirroring performed by the database itself, except that the database system does not have to bother about the details of performing the mirroring. It just issues the write to the RAID system, which automatically performs the mirroring.

RAID level 5 is less expensive than mirroring in terms of disk space requirement, but writes are more expensive, and rebuilding a crashed disk is more expensive.

CHAPTER 21

Application Development and Administration

Exercises

21.1 What is the main reason why servlets give better performance than programs that use the common gateway interface (CGI), even though Java programs generally run slower than C or C++ programs.

Answer: The CGI interface starts a new process to service each request, which has a significant operating system overhead. On the other hand, servelets are run as threads of an existing process, avoiding this overhead. Further, the process running threads could be the Web server process itself, avoiding interprocess communication which can be expensive. Thus, for small to moderate sized tasks, the overhead of Java is less than the overheads saved by avoiding process creating and communication.

For tasks involving a lot of CPU activity, this may not be the case, and using CGI with a C or C++ program may give better performance.

21.2 List some benefits and drawbacks of connectionless protocols over protocols that maintain connections.

Answer: Most computers have limits on the number of simultaneous connections they can accept. With connectionless protocols, connections are broken as soon as the request is satisfied, and therefore other clients can open connections. Thus more clients can be served at the same time. A request can be routed to any one of a number of different servers to balance load, and if a server crashes another can take over without the client noticing any problem.

The drawback of connectionless protocols is that a connection has to be reestablished every time a request is sent. Also, session information has to be sent each time in form of cookies or hidden fields. This makes them slower than the protocols which maintain connections in case state information is required.

21.3 List three ways in which caching can be used to speed up Web server performance.

Answer: Caching can be used to improve performance by exploiting the commonalities between transactions.

a. If the application code for servicing each request needs to open a connection to the database, which is time consuming, then a pool of open connections may be created before hand, and each request uses one from those.

b. The results of a query generated by a request can be cached. If same request comes agian, or generates the same query, then the cached result can be used instead of connecting to database again.

c. The final webpage generated in response to a request can be cached. If the same request comes again, then the cached page can be outputed.

21.4 **a.** What are the three broad levels at which a database system can be tuned to improve performance?

b. Give two examples of how tuning can be done, for each of the levels.

Answer:

a. We refer to performance tuning of a database system as the modification of some system components in order to improve transaction response times, or overall transaction throughput. Database systems can be tuned at various levels to enhance performance. viz.

i. Schema and transaction design
ii. Buffer manager and transaction manager
iii. Access and storage structures
iv. Hardware - disks, CPU, busses etc.

b. We describe some examples for performance tuning of some of the major components of the database system.

i. Tuning the schema -

In this chapter we have seen two examples of schema tuning, viz. vertical partition of a relation (or conversely - join of two relations), and denormalization (or conversely - normalization). These examples reflect the general scenario, and ideas therein can be applied to tune other schemas.

ii. Tuning the transactions -

One approach used to speed-up query execution is to improve the its plan. Suppose that we need the natural join of two relations - say *account* and *depositor* from our sample bank database. A *sort–merge-join*(Section 13.5.4) on the attribute *account-number* may be quicker than a simple nested-loop join on the relations.

Other ways of tuning transactions are - breaking up long update transactions and combining related sets of queries into a single query. Generic examples for these approaches are given in this chapter.

For client-server systems, wherein the query has to be transmitted from client to server, the query transmission time itself may form a

large fraction of the total query cost. Using *stored procedures* can significantly reduce the queries response time.

iii. Tuning the buffer manager -

The buffer manager can be made to increase or decrease the number of pages in the buffer according to changing page-fault rates. However, it must be noted that a larger number of pages may mean higher costs for latch management and maintenance of other data-structures like free-lists and page map tables.

iv. Tuning the transaction manager -

The transaction schedule affects system performance. A query that computes statistics for customers at each branch of the bank will need to scan the relations *account* and *depositor*. During these scans, no updates to any customer's balance will be allowed. Thus, the response time for the update transactions is high. Large queries are best executed when there are few updates, such as at night.

Checkpointing also incurs some cost. If recovery time is not critical, it is preferable to examine a long log (during recovery) rather than spend a lot of (checkpointing) time during normal operation. Hence it may be worthwhile to tune the checkpointing interval according to the expected rate of crashes and the required recovery time.

v. Tuning the access and storage structures -

A query's response time can be improved by creating an appropriate index on the relation. For example, consider a query in which a depositor enquires about her balance in a particular account. This query would result in the scan of the relation *account* if it has is no index on *account-number*. Similar indexing considerations also apply to computing joins. i.e an index on *account-number* in the *account* relation saves scanning *account* when a natural join of *account* is taken with *depositor*.

In contrast, performance of update transactions may suffer due to indexing. Let us assume that frequent updates to the balance are required. Also suppose that there is an index on *balance* (presumably for range queries) in *account*. Now, for each update to the value of the balance, the index too will have to be updated. In addition, concurrent updates to the index structure will require additional locking overheads. Note that the response time for each update would not be more if there were no index on *balance*.

The type of index chosen also affects performance. For a range query, an order preserving index (like B-trees) is better than a hashed index.

Clustering of data affects the response time for some queries. For example, assume that the tuples of the *account* relation are clustered on *branch-name*. Then the average execution time for a query that finds the total balance amount deposited at a particular branch can be improved. Even more benefit accrues from having a clustered index on *branch-name*.

If the database system has more than one disk, *declustering* of data will enable parallel access. Suppose that we have five disks and that

in a hypothetical situation where each customer has five accounts and each account has a lot of historical information that needs to be accessed. Storing one account per customer per disk will enable parallel access to all accounts of a particular customer. Thus, the speed of a scan on *depositor* will increase about five-fold.

vi. Tuning the hardware -

The hardware for the database system typically consists of disks, the processor, and the interconnecting architecture (busses etc.). Each of these components may be a bottleneck and by increasing the number of disks or their block-sizes, or using a faster processor, or by improving the bus architecture, one may obtain an improvement in system performance.

21.5 What is the motivation for splitting a long transaction into a series of small ones? What problems could arise as a result, and how can these problems be averted?

Answer: Long update transactions cause a lot of log information to be written, and hence extend the checkpointing interval and also the recovery time after a crash. A transaction that performs many updates may even cause the system log to overflow before the transaction commits.

To avoid these problems with a long update transaction it may be advisable to break it up into smaller transactions. This can be seen as a *group* transaction being split into many small *mini-batch* transactions. The same effect is obtained by executing both the group transaction and the mini-batch transactions, which are scheduled in the order that their operations appear in the group transaction.

However, executing the mini-batch transactions in place of the group transaction has some costs, such as extra effort when recovering from system failures. Also, even if the group transaction satisfies the *isolation* requirement, the mini-batch may not. Thus the transaction manager can release the locks held by the mini-batch only when the last transaction in the mini-batch completes execution.

21.6 Suppose a system runs three types of transactions. Transactions of type A run at the rate of 50 per second, transactions of type B run at 100 per second, and transactions of type C run at 200 per second. Suppose the mix of transactions has 25 percent of type A, 25 percent of type B, and 50 percent of type C.

a. What is the average transaction throughput of the system, assuming there is no interference between the transactions.

b. What factors may result in interference between the transactions of different types, leading to the calculated throughput being incorrect?

Answer:

a. Let there be 100 transactions in the system. The given mix of transaction types would have 25 transactions each of type A and B, and 50 transactions of type C. Thus the time taken to execute transactions only of type A is 0.5

seconds and that for transactions only of type B or only of type C is 0.25 seconds. Given that the transactions do not interfere, the total time taken to execute the 100 transactions is $0.5 + 0.25 + 0.25 = 1$ second. i.e, the average overall transaction throughput is 100 *transactions per second*.

b. One of the most important causes of transaction interference is lock contention. In the previous example, assume that transactions of type A and B are update transactions, and that those of type C are queries. Due to the speed mismatch between the processor and the disk, it is possible that a transaction of type A is holding a lock on a "hot" item of data and waiting for a disk write to complete, while another transaction (possibly of type B or C) is waiting for the lock to be released by A. In this scenario some CPU cycles are wasted. Hence, the observed throughput would be lower than the calculated throughput.

Conversely, if transactions of type A and type B are disk bound, and those of type C are CPU bound, and there is no lock contention, observed throughput may even be better than calculated.

Lock contention can also lead to deadlocks, in which case some transaction(s) will have to be aborted. Transaction aborts and restarts (which may also be used by an optimistic concurrency control scheme) contribute to the observed throughput being lower than the calculated throughput.

Factors such as the limits on the sizes of data-structures and the variance in the time taken by book-keeping functions of the transaction manager may also cause a difference in the values of the observed and calculated throughput.

21.7 Suppose the price of memory falls by half, and the speed of disk access (number of accesses per second) doubles, while all other factors remain the same. What would be the effect of this change on the 5 minute and 1 minute rule?

Answer: There will be no effect of these changes on the 5 minute or the 1 minute rule. The value of n, i.e. the frequency of page access at the break-even point, is proportional to the product of memory price and speed of disk access, other factors remaining constant. So when memory price falls by half and access speed doubles, n remains the same.

21.8 List some of the features of the TPC benchmarks that help make them realistic and dependable measures.

Answer: Some features that make the TPC benchmarks realistic and dependable are -

a. Ensuring full support for ACID properties of transactions,
b. Calculating the throughput by observing the *end-to-end* performance,
c. Making sizes of relations proportional to the expected rate of transaction arrival, and
d. Measuring the dollar cost per unit of throughput.

21.9 Why was the TPCD benchmark replaced by the TPCH and TPCR benchmarks?

Answer: Various TPCD queries can be significantly speeded up by using materialized views and other redundant information, but the overheads of using

them should be properly accounted. Hence TPCR and TPCH were introduced as refinements of TPCD, both of which use same schema and workload. TPCR models periodic reporting queries, and the database running it is permited to use materialized views. TPCH, on the other hand, models ad hoc querying, and prohibits materialized views and other redundant information.

21.10 List some benefits and drawbacks of an anticipatory standard compared to a reactionary standard.

Answer: In the absence of an anticipatory standard it may be difficult to reconcile between the differences among products developed by various organizations. Thus it may be hard to formulate a reactionary standard without sacrificing any of the product development effort. This problem has been faced while standardizing pointer syntax and access mechanisms for the ODMG standard.

On the other hand, a reactionary standard is usually formed after extensive product usage, and hence has an advantage over an anticipatory standard - that of built-in pragmatic experience. In practice, it has been found that some anticipatory standards tend to be over-ambitious. SQL-3 is an example of a standard that is complex and has a very large number of features. Some of these features may not be implemented for a long time on any system, and some, no doubt, will be found to be inappropriate.

21.11 Suppose someone impersonates a company and gets a certificate from a certificate issuing authority. What is the effect on things (such as purchase orders or programs) certified by the impersonated company, and on things certified by other companies?

Answer: The key problem with digital certificates (when used offline, without contacting the certificate issuer) is that there is no way to withdraw them.

For instance (this actually happened, but names of the parties have been changed) person C claims to be an employee of company X and get a new public key certified by the certifying authority A. Suppose the authority A incorrectly believed that C was acting on behalf of company X, it gives C a certificate $cert$. Now, C can communicate with person Y, who checks the certificate $cert$ presenetd by C, and believes the public key contained in $cert$ really belongs to X. Now C would communicate with Y using the public key, and Y trusts the communication is from company X.

Person Y may now reveal confidential information to C, or accept purchase order from C, or execute programs certified by C, based on the public key, thinking he is actually communicating with company X. In each case there is potential for harm to Y.

Even if A detects the impersonation, as long as Y does not check with A (the protocol does not require this check), there is no way for Y to find out that the certificate is forged.

If X was a certification authority itself, further levels of fake certificates can be created. But certificates that are not part of this chain would not be affected.

CHAPTER 22

Advanced Querying and Information Retrieval

This chapter covers advanced querying techniques for databases and information retrieval. Advanced querying techniques include decision support systems, online analytical processing, including SQL:1999 support for OLAP, and data mining.

Although information retrieval has been considered as a separate field from databases in the research community, there are strong connections. Distributed information retrieval is growing in importance with the explosion of documents on the world wide web and the resultant importance of web search techniques.

Considering the growing importance of all the topics covered in this chapter, some of the sections of the chapter can be assigned as supplementary reading material, even in an introductory course. These could include OLAP, some parts of data mining, and some parts of information retrieval. The material in the chapter is also suitable for laying the groundwork for an advanced course, or for professionals to keep in touch with recent developments.

Changes from 3rd edition:

- Coverage of OLAP has been extended with coverage of hierarchies, and new material on OLAP support in SQL:1999 has been introduced, including extended aggregation, ranking and windowing.
- The section on data mining has been significantly extended, with new material on different types of mining, including classification, associations and clustering, and different approaches to classification and regression. We have also introduced coverage of algorithms for decision tree construction and for finding association rules.
- Coverage of data warehouses has been extended, with coverage of star schemas.

- Coverage of information retrieval has been extended, with better coverage of basic information retrieval, and coverage of information retrieval on the Web, exploiting hyperlink information.

Exercises

22.1 For each of the SQL aggregate functions **sum, count, min** and **max**, show how to compute the aggregate value on a multiset $S_1 \cup S_2$, given the aggregate values on multisets S_1 and S_2.

Based on the above, give expressions to compute aggregate values with grouping on a subset S of the attributes of a relation $r(A, B, C, D, E)$, given aggregate values for grouping on attributes $T \supseteq S$, for the following aggregate functions:

a. **sum, count, min** and **max**
b. **avg**
c. standard deviation

Answer: Given aggregate values on multisets S_1 and S_2, e can calulate the corresponding aggregate values on multiset $S_1 \cup S_2$ as follows:

a. **sum**$(S_1 \cup S_2) =$ **sum**$(S_1) +$ **sum**(S_2)
b. **count**$(S_1 \cup S_2) =$ **count**$(S_1) +$ **count**(S_2)
c. **min**$(S_1 \cup S_2) =$ **min**(**min**(S_1), **min**(S_2))
d. **max**$(S_1 \cup S_2) =$ **max**(**max**(S_1), **max**(S_2))

Let the attribute set *T = (A, B, C, D)* and the attribute set *S = (A, B)*. Let the aggregation on the attribute set T be stored in table *aggregation-on-t* with aggregation columns *sum-t*, *count-t*, *min-t*, and *max-t* storing **sum, count, min** and **max** resp.

a. The aggregations *sum-s*, *count-s*, *min-s*, and *max-s* on the attribute set *S* are computed by the query:

```
select A, B, sum(sum-t) as sum-s, sum(count-t) as count-s,
             min(min-t) as min-s, max(max-t) as max-s
from aggregation-on-t
groupby A, B
```

b. The aggregation *avg* on the attribute set *S* is computed by the query:

```
select A, B, sum(sum-t)/sum(count-t) as avg-s
from aggregation-on-t
groupby A, B
```

c. For calculating standard deviation we use an alternative formula:

$$stddev(S) = \frac{\sum_{s \in S} s^2}{|S|} - avg(S)^2$$

which we get by expanding the formula

$$stddev(S) = \frac{\sum_{s \in S} (s^2 - avg(S))^2}{|S|}$$

If S is partitioned into n sets $S_1, S_2, \ldots S_n$ then the following relation holds:

$$stddev(S) = \frac{\sum_{S_i} |S_i|(stddev(S_i)^2 + avg(S_i)^2)}{|S|} - avg(S)^2$$

Using this formula, the aggregation **stddev** is computed by the query:

select *A*, *B*,
    [**sum**(*count-t* $*$ (*stddev-t*^{2}+ *avg-t*^{2}))/**sum**(*count-t*)] -
        [**sum**(*sum-t*)/**sum**(*count-t*)]
from *aggregation-on-t*
groupby *A*, *B*

22.2 Show how to express **group by cube**(a, b, c, d) using **rollup**; your answer should have only one **group by** clause.
Answer:

groupby rollup(*a*), **rollup**(*b*), **rollup**(*c*), **rollup**(*d*)

22.3 Give an example of a pair of groupings that cannot be expressed by using a single **group by** clause with **cube** and **rollup**.
Answer: Consider an example of hierarchies on dimensions from Figure 22.4. We can not express a query to seek aggregation on groups (*City*, *Hour of day*) and (*City*, *Date*) using a single **group by** clause with **cube** and **rollup**.

Any single **groupby** clause with **cube** and **rollup** that computes these two groups would also compute other groups also.

22.4 Given a relation $S(student, subject, marks)$, write a query to find the top n students by total marks, by using ranking.
Answer: We assume that multiple students do not have the same marks since otherwise the question is not deterministic; the query below deterministically returns all students with the same marks as the n student, so it may return more than n students.

select *student*, **sum**(*marks*) **as** *total*,
        rank() **over** (**order by** (*total*) **desc**) **as** *trank*
from *S*
groupby *student*
having *trank* $\leq n$

22.5 Given relation $r(a, b, d, d)$, Show how to use the extended SQL features to generate a histogram of d versus a, dividing a into 20 equal-sized partitions (that is, where each partition contains 5 percent of the tuples in r, sorted by a).
Answer:

```
select tile20, sum(d)
from (select d, ntile(20) over (order by (a)) as tile20
      from r) as s
groupby tile20
```

22.6 Write a query to find cumulative balances, equivalent to that shown in Section 22.2.5, but without using the extended SQL windowing constructs.
Answer:

```
select t1.account-number, t1.date-time, sum(t2.value)
from transaction as t1, transaction as t2
where t1.account-number = t2.account-number and
        t2.date-time < t1.date-time
groupby t1.account-number, t1.date-time
order by t1.account-number, t1.date-time
```

22.7 Consider the *balance* attribute of the *account* relation. Write an SQL query to compute a histogram of *balance* values, dividing the range 0 to the maximum account balance present, into three equal ranges.
Answer:

```
(select 1, count(*)
 from account
 where 3* balance <= (select max(balance)
                      from account)
)
union
(select 2, count(*)
 from account
 where 3* balance > (select max(balance)
                     from account)
       and 1.5* balance <= (select max(balance)
                            from account)
)
union
(select 3, count(*)
 from account
 where 1.5* balance > (select max(balance)
                       from account)
)
```

22.8 Consider the *sales* relation from Section 22.2. Write an SQL query to compute the cube operation on the relation, giving the relation in Figure 22.2. Do not use the **with cube** construct.
Answer:

```
(select color, size, sum(number)
 from sales
 groupby color, size
)
union
(select color, 'all', sum(number)
 from sales
 groupby color
)
union
(select 'all', size, sum(number)
 from sales
 groupby size
)
union
(select 'all', 'all', sum(number)
 from sales
)
```

22.9 Construct a decision tree classifier with binary splits at each node, using tuples in relation $r(A, B, C)$ shown below as training data; attribute C denotes the class. Show the final tree, and with each node show the best split for each attribute along with its information gain value.

$$(1, 2, a), (2, 1, a), (2, 5, b), (3, 3, b), (3, 6, b), (4, 5, b), (5, 5, c), (6, 3, b), (6, 7, c)$$

Answer:

22.10 Suppose there are two classification rules, one that says that people with salaries between \$10,000 and \$20,000 have a credit rating of *good*, and another that says that people with salaries between \$20,000 and \$30,000 have a credit rating of *good*. Under what conditions can the rules be replaced, without any loss of information, by a single rule that says people with salaries between \$10,000 and \$30,000 have a credit rating of *good*.

Answer: Consider the following pair of rules and their confidence levels :

No.	Rule	Conf.
1.	$\forall\ persons\ \ P,\ 10000 < P.salary \leq 20000 \Rightarrow P.credit = good$	60%
2.	$\forall\ persons\ \ P,\ 20000 < P.salary \leq 30000 \Rightarrow P.credit = good$	90%

The new rule has to be assigned a confidence-level which is between the confidence-levels for rules 1 and 2. Replacing the original rules by the new rule will result in a loss of confidence-level information for classifying persons, since we cannot distinguish the confidence levels of perople earning between 10000 and 20000 from those of people earning between 20000 and 30000. There-

fore we can combine the two rules without loss of information only if their confidences are the same.

22.11 Suppose half of all the transactions in a clothes shop purchase jeans, and one third of all transactions in the shop purchase T-shirts. Suppose also that half of the transactions that purchase jeans also purchase T-shirts. Write down all the (nontrivial) association rules you can deduce from the above information, giving support and confidence of each rule.
Answer: The rules are as follows. The last rule can be deduced from the previous ones.

Rule	Support	Conf.
$\forall\, transactions\ T,\ \mathbf{true} \Rightarrow buys(T, jeans)$	50%	50%
$\forall\, transactions\ T,\ \mathbf{true} \Rightarrow buys(T, t\text{-}shirts)$	33%	33%
$\forall\, transactions\ T,\ buys(T, jeans) \Rightarrow buys(T, t\text{-}shirts)$	25%	50%
$\forall\, transactions\ T,\ buys(T, t\text{-}shirts) \Rightarrow buys(T, jeans)$	25%	75%

22.12 Consider the problem of finding large itemsets.

a. Describe how to find the support for a given collection of itemsets by using a single scan of the data. Assume that the itemsets and associated information, such as counts, will fit in memory.

b. Suppose an itemset has support less than j. Show that no superset of this itemset can have support greater than or equal to j.

Answer:

a. Let $\{S_1, S_2, \ldots, S_n\}$ be the collection of item-sets for which we want to find the support. Associate a counter $count(S_i)$ with each item-set S_i.

Initialize each counter to zero. Now examine the transactions one-by-one. Let $S(T)$ be the item-set for a transaction T. For each item-set S_i that is a subset of $S(T)$, increment the corresponding counter $count(S_i)$.

When all the transactions have been scanned, the values of $count(S_i)$ for each i will give the support for item-set S_i.

b. Let A be an item-set. Consider any item-set B which is a superset of A. Let τ_A and τ_B be the sets of transactions that purchase all items in A and all items in B, respectively. For example, suppose A is $\{a, b, c\}$, and B is $\{a, b, c, d\}$.

A transaction that purchases all items from B must also have purchased all items from A (since $A \subseteq B$). Thus, every transaction in τ_B is also in τ_A. This implies that the number of transactions in τ_B is at most the number of transactions in τ_A. In other words, the support for B is at most the support for A.

Thus, if any item-set has support less than j, all supersets of this item-set have support less than j.

22.13 Describe benefits and drawbacks of a source-driven architecture for gathering of data at a data-warehouse, as compared to a destination-driven architecture.

Answer: In a destination-driven architecture for gathering data, data transfers from the data sources to the data-warehouse are based on demand from the warehouse, whereas in a source-driven architecture, the transfers are initiated by each source.

The benefits of a source-driven architecture are

- Data can be propagated to the destination as soon as it becomes available. For a destination-driven architecture to collect data as soon as it is available, the warehouse would have to probe the sources frequently, leading to a high overhead.
- The source does not have to keep historical information. As soon as data is updated, the source can send an update message to the destination and forget the history of the updates. In contrast, in a destination-driven architecture, each source has to maintain a history of data which have not yet been collected by the data warehouse. Thus storage requirements at the source are lower for a source-driven architecture.

On the other hand, a destination-driven architecture has the following advantages.

- In a source-driven architecture, the source has to be active and must handle error conditions such as not being able to contact the warehouse for some time. It is easier to implement passive sources, and a single active warehouse. In a destination-driven architecure, each source is required to provide only a basic functionality of executing queries.
- The warehouse has more control on when to carry out data gathering activities, and when to process user queries; it is not a good idea to perform both simultaneously, since they may conflict on locks.

22.14 Consider the schema depicted in Figure 22.9. Give an SQL:1999 query to summarize sales numbers and price by store and date, along with the hierarchies on store and date.

Answer:

```
select store-id, city, state, country,
       date, month, quarter, year,
       sum(number), sum(price)
from sales, store, date
where sales.store-id = store.store-id and
       sales.date = date.date
groupby rollup(country, state, city, store-id),
       rollup(year, quarter, month, date)
```

22.15 Compute the relevance (using appropriate definitions of term frequency and inverse document frequency) of each of the questions in this chapter to the query "SQL relation."

Answer: We do not consider the questions containing neither of the keywords as their relevance to the keywords is zero. The number of words in a question

include stop words. We use the equations given in Section 22.5.1.1 to compute relevance; the log term in the equation is assumed to be to the base 2.

Q#	#wo--rds	# "SQL"	#"rela--tion"	"SQL" term freq.	"relation" term freq.	"SQL" relv.	"relation" relv.	Total relv.
1	84	1	1	0.0170	0.0170	0.0002	0.0002	0.0004
4	22	0	1	0.0000	0.0641	0.0000	0.0029	0.0029
5	46	1	1	0.0310	0.0310	0.0006	0.0006	0.0013
6	22	1	0	0.0641	0.0000	0.0029	0.0000	0.0029
7	33	1	1	0.0430	0.0430	0.0013	0.0013	0.0026
8	32	1	3	0.0443	0.1292	0.0013	0.0040	0.0054
9	77	0	1	0.0000	0.0186	0.0000	0.0002	0.0002
14	30	1	0	0.0473	0.0000	0.0015	0.0000	0.0015
15	26	1	1	0.0544	0.0544	0.0020	0.0020	0.0041

22.16 What is the difference between a false positive and a false drop? If it is essential that no relevant information be missed by an information retrieval query, is it acceptable to have either false positives or false drops? Why?

Answer: Information-retrieval systems locate documents that contain a specified keyword by using an index that maps this keyword onto a set of identifiers for documents containing it. Each keyword may be contained in a large number of documents. To save on storage space for the document identifiers corresponding to a keyword, the index is sometimes stored such that the retrieval is *approximate*. The error in this approximation may lead to one of two situations – a *false drop* occurs when some relevant document is not retrieved; and a *false positive* occurs when some irrelevant document is retrieved. Thus, for information-retrieval queries that mandate no loss of relevant information, it is acceptable to have false positives, but not false drops.

22.17 Suppose you want to find documents that contain at least k of a given set of n keywords. Suppose also you have a keyword index that gives you a (sorted) list of identifiers of documents that contain a specified keyword. Give an efficient algorithm to find the desired set of documents.

Answer: Let S be a set of n keywords. An algorithm to find all documents that contain at least k of these keywords is given below :

This algorithm calculates a reference count for each document identifier. A reference count of i for a document identifier d means that at least i of the keywords in S occur in the document identified by d. The algorithm maintains a list of records, each having two fields – a document identifier, and the refer-

ence count for this identifier. This list is maintained sorted on the document identifier field.

```
initialize the list L to the empty list;
for (each keyword c in S) do
begin
   D := the list of documents identifiers corresponding to c;
   for (each document identifier d in D) do
      if (a record R with document identifier as d is on list L) then
            R.reference_count := R.reference_count + 1;
      else begin
            make a new record R;
            R.document_id := d;
            R.reference_count := 1;
            add R to L;
      end;
end;
for (each record R in L) do
   if (R.reference_count >= k) then
      output R;
```

Note that execution of the second *for* statement causes the list D to "merge" with the list L. Since the lists L and D are sorted, the time taken for this merge is proportional to the sum of the lengths of the two lists. Thus the algorithm runs in time (at most) proportional to n times the sum total of the number of document identifiers corresponding to each keyword in S.

CHAPTER 23

Advanced Data Types and New Applications

This chapter covers advanced data types and new applications, including temporal datababases, spatial and geographic databases, multimedia databases, and mobility and personal databases. In particular, the data types mentioned above have grown in importance in recent years, and commercial database systems are increasingly providing support for such data types through extensions to the database system variously called cartridges or extenders.

This chapter is suited as a means to lay the groundwork for an advanced course. Some of the material, such as temporal and spatial data types, may be suitable for self-study in a first course.

Changes from 3rd edition:
This material was part of Chapter 21 in the previous edition, but that chapter has been split into two chapters, Chapters 22 and 23, in this edition.

Coverage of R-trees has been extended, with an informal description of insertion and deletion algorithms. Coverage of mobile data communication has been updated.

Exercises

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

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

Suppose we consider our sample bank database to be bitemporal. Only the concept of valid time allows the system to answer queries such as - "What was

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

23.2 Will functional dependencies be preserved if a relation is converted to a temporal relation by adding a time attribute? How is the problem handled in a temporal database?

Answer: Functional dependencies may be violated when a relation is augmented to include a time attribute. For example, suppose we add a time attribute to the relation *account* in our sample bank database. The dependency *account-number→balance* may be violated since a customer's balance would keep changing with time.

To remedy this problem temporal database systems have a slightly different notion of functional dependency, called *temporal functional dependency*. For example, the temporal functional dependency *account-number* $\stackrel{T}{\rightarrow}$ *balance* over *Account-schema* means that for each instance *account* of *Account-schema*, all snapshots of *account* satisfy the functional dependency *account-number→balance*; i.e at any time instance, each account will have a unique bank balance corresponding to it.

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

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

23.4 Consider two-dimensional vector data where the data items do not overlap. Is it possible to convert such vector data to raster data? If so, what are the drawbacks of storing raster data obtained by such conversion, instead of the original vector data?

Answer: To convert non-overlapping vector data to raster data, we set the values for exactly those pixels that lie on any one of the data items (regions); the other pixels have a default value.

The disadvantages to this approach are: loss of precision in location information (since raster data loses resolution), a much higher storage requirement, and loss of abstract information (like the shape of a region).

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

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

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

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

Answer:

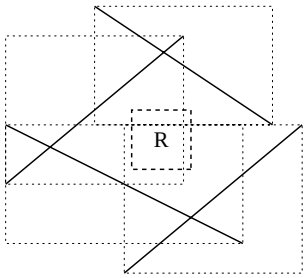


Figure 21.17 a) : Representation of a Segment by One Rectangle

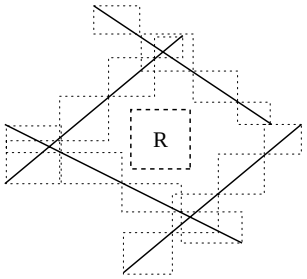


Figure 21.17 b) : Splitting each Segment into Four Pieces

Large bounding boxes tend to overlap even where the region of overlap does not contain any information. The Figure 21.17 a) shows a region R within which we have to locate a segment. Note that even though none of the four segments lies in R, due to the large bounding boxes, we have to check each of the four bounding boxes to confirm this. A significant improvement is ob-

served in the Figure 21.17 b), where each segment is split into multiple pieces, each with its own bounding box. In the second case, the box R is not part of the boxes indexed by the R-tree. In general, dividing a segment into smaller pieces causes the bounding boxes to be smaller and less wasteful of area.

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

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

23.8 Study the support for spatial data offered by the database system that you use, and implement the following:

a. A schema to represent the geographic location of restaurants along with features such as the cuisine served at the restaurant and the level of expensiveness.

b. A query to find moderately priced restaurants that serve Indian food and are within 5 miles of your house (assume any location for your house).

c. A query to find for each restaurant the distance from the nearest restaurant serving the same cuisine and with the same level of expensiveness.

Answer: TO BE SOLVED

23.9 What problems can occur in a continuous-media system if data is delivered either too slowly or too fast?

Answer: Continuous media systems typically handle a large amount of data, which have to be delivered at a steady rate. Suppose the system provides the picture frames for a television set. The delivery rate of data from the system should be matched with the frame display rate of the TV set. If the delivery rate is too low, the display would periodically freeze or blank out, since there will be no new data to be displayed for some time. On the other hand, if the delivery rate is too high, the data buffer at the destination TV set will overflow causing loss of data; the lost data will never get displayed.

23.10 Describe how the ideas behind the RAID organization (Section 11.3) can be used in a broadcast-data environment, where there may occasionally be noise that prevents reception of part of the data being transmitted.

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

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

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

23.11 List three main features of mobile computing over wireless networks that are distinct from traditional distributed systems.

Answer: Some of the main distinguishing features are as follows.

- In distributed systems, disconnection of a host from the network is considered to be a *failure*, whereas allowing such disconnection is a *feature* of mobile systems.
- Distributed systems are usually centrally administered, whereas in mobile computing, each personal computer that participates in the system is administered by the user (owner) of the machine and there is little central administration, if any.
- In conventional distributed systems, each machine has a fixed location and network address(es). This is not true for mobile computers, and in fact, is antithetical to the very purpose of mobile computing.
- Queries made on a mobile computing system may involve the location and velocity of a host computer.
- Each computer in a distributed system is allowed to be arbitrarily large and may consume a lot of (almost) uninterrupted electrical power. Mobile

systems typically have small computers that run on low wattage, short-lived batteries.

23.12 List three factors that need to be considered in query optimization for mobile computing that are not considered in traditional query optimizers.
Answer: The most important factor influencing the cost of query processing in traditional database systems is that of disk I/O. However, in mobile computing, minimizing the amount of energy required to execute a query is an important task of a query optimizer. To reduce the consumption of energy (battery power), the query optimizer on a mobile computer must minimize the size and number of queries to be transmitted to remote computers as well as the time for which the disk is spinning.

In traditional database systems, the cost model typically does not include connection time and the amount of data transferred. However, mobile computer users are usually charged according to these parameters. Thus, these parameters should also be minimized by a mobile computer's query optimizer.

23.13 Define a model of repeatedly broadcast data in which the broadcast medium is modeled as a virtual disk. Describe how access time and data-transfer rate for this virtual disk differ from the corresponding values for a typical hard disk.
Answer: We can distinguish two models of broadcast data. In the case of a pure broadcast medium, where the receiver cannot communicate with the broadcaster, the broadcaster transmits data with periodic cycles of retransmission of the entire data, so that new receivers can catch up with all the broadcast information. Thus, the data is broadcast in a continuous cycle. This period of the cycle can be considered akin to the worst case rotational latency in a disk drive. There is no concept of seek time here. The value for the cycle latency depends on the application, but is likely to be at least of the order of seconds, which is much higher than the latency in a disk drive.

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

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

23.14 Consider a database of documents in which all documents are kept in a central database. Copies of some documents are kept on mobile computers. Suppose that mobile computer A updates a copy of document 1 while it is disconnected,

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

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

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

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

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

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

23.15 Give an example to show that the version-vector scheme does not ensure serializability. (Hint: Use the example from Exercise 23.14, with the assumption that documents 1 and 2 are available on both mobile computers A and B, and take into account the possibility that a document may be read without being updated.)

Answer: Consider the example given in the previous exercise. Suppose that both host A and host B are not connected to each other. Further, assume that identical copies of document 1 and document 2 are stored at host A and host B.

Let $\{X = 5\}$ be the initial contents of document 1, and $\{X = 10\}$ be the initial contents of document 2. Without loss of generality, let us assume that all version-vectors are initially zero.

Suppose host A updates the number its copy of document 1 with that in its copy of document 2. Thus, the contents of both the documents (at host A) are now $\{X = 10\}$. The version number $V_{1,A,A}$ is incremented to 1.

While host B is disconnected from host A, it updates the number in its copy of document 2 with that in its copy of document 1. Thus, the contents of both the documents (at host B) are now $\{X = 5\}$. The version number $V_{2,B,B}$ is incremented to 1.

Later, when host A and host B connect, they exchange version-vectors. The version-vector scheme updates the copy of document 1 at host B to $\{X = 10\}$, and the copy of document 2 at host A to $\{X = 5\}$. Thus, both copies of each document are identical, viz. document 1 contains $\{X = 10\}$ and document 2 contains $\{X = 5\}$.

However, note that a serial schedule for the two updates (one at host A and another at host B) would result in both documents having the *same* contents. Hence this example shows that the version-vector scheme does not ensure serializability.

CHAPTER 24

Advanced Transaction Processing

In this chapter, we go beyond the basic transaction processing schemes discussed previously, and cover more advanced transaction-processing concepts, including transaction-processing monitors, workflow systems, main-memory databases, real-time transaction systems, and handling of long-duration transactions by means of nested transactions, multi-level transactions and weak degrees of consistency. We end the chapter by covering weak degrees of consistency used to handle multidatabase systems.

This chapter is suited to an advanced course. The sections on TP monitors and workflows may also be covered in an introductory course as independent-study material.

Changes from 3rd edition:
Coverage of remote backup systems has been moved from this chapter to the chapter on recovery, while coverage of transaction processing in multidatabases has been moved into this chapter from its earlier position in the distributed database chapter.

Exercises

24.1 Explain how a TP monitor manages memory and processor resources more effectively than a typical operating system.

Answer: In a typical OS, each client is represented by a process, which occupies a lot of memory. Also process multi-tasking over-head is high.

A TP monitor is more of a service provider, rather than an environment for executing client processes. The client processes run at their own sites, and they send requests to the TP monitor whenever they wish to avail of some service. The message is routed to the right server by the TP monitor, and the results of the service are sent back to the client.

The advantage of this scheme is that the same server process can be serving several clients simultaneously, by using multithreading. This saves memory space, and reduces CPU overheads on preserving ACID properties and on scheduling entire processes. Even without multi-threading, the TP monitor can dynamically change the number of servers running, depending on whatever factors affect good performance. All this is not possible with a typical OS setup.

24.2 Compare TP monitor features with those provided by Web servers supporting servlets (such servers have been nicknamed *TP-lite*).
Answer: TO BE FILLED IN.

24.3 Consider the process of admitting new students at your university (or new employees at your organization).

a. Give a high-level picture of the workflow starting from the student application procedure.
b. Indicate acceptable termination states, and which steps involve human intervention.
c. Indicate possible errors (including deadline expiry) and how they are dealt with.
d. Study how much of the workflow has been automated at your university.

Answer: TO BE FILLED IN.

24.4 Like database systems, workflow systems also require concurrency and recovery management. List three reasons why we cannot simply apply a relational database system using 2PL, physical undo logging, and 2PC.
Answer:

a. The tasks in a workflow have dependencies based on their status. For example the starting of a task may be conditional on the outcome (such as commit or abort) of some other task. All the tasks cannot execute independently and concurrently, using 2PC just for atomic commit.
b. Once a task gets over, it will have to expose its updates, so that other tasks running on the same processing entity don't have to wait for long. 2PL is too strict a form of concurrency control, and is not appropriate for workflows.
c. Workflows have their own consistency requirements, i.e. failure-atomicity. An execution of a workflow must finish in an *acceptable termination state*. Because of this, and because of early exposure of uncommitted updates, the recovery procedure will be quite different. Some form of logical logging and compensation transactions will have to be used. Also to perform forward recovery of a failed workflow, the recovery routines need to restore the state information of the scheduler and tasks, not just the updated data items. Thus simple WAL cannot be used.

24.5 If the entire database fits in main memory, do we still need a database system to manage the data? Explain your answer.

Answer: Even if the entire database fits in main memory, a DBMS is needed to perform tasks like concurrency control, recovery, logging etc, in order to preserve ACID properties of transactions.

24.6 Consider a main-memory database system recovering from a system crash. Explain the relative merits of

- Loading the entire database back into main memory before resuming transaction processing
- Loading data as it is requested by transactions

Answer:

- Loading the entire database into memory in advance can provide transactions which need high-speed or realtime data access the guarantee that once they start they will not have to wait for disk accesses to fetch data. However no transaction can run till the entire database is loaded.
- The advantage in loading on demand is that transaction processing can start rightaway; however transactions may see long and unpredictable delays in disk access until the entire database is loaded into memory.

24.7 In the group-commit technique, how many transactions should be part of a group? Explain your answer.
Answer: As log-records are written to stable storage in multiples of a block, we should group transaction commits in such a way that the last block containing log-records for the current group is almost full.

24.8 Is a high-performance transaction system necessarily a real-time system? Why or why not?
Answer: A high-performance system is not necessarily a real-time system. In a high performance system, the main aim is to execute each transaction as quickly as possible, by having more resources and better utilization. Thus average speed and response time are the main things to be optimized. In a real-time system, speed is not the central issue. Here *each* transaction has a deadline, and taking care that it finishes within the deadline or takes as little extra time as possible, is the critical issue.

24.9 In a database system using write-ahead logging, what is the worst-case number of disk accesses required to read a data item? Explain why this presents a problem to designers of real-time database systems.
Answer: In the worst case, a read can cause a buffer page to be written to disk (preceded by the corresponding log records), followed by the reading from disk of the page containing the data to be accessed. This takes two or more disk accesses, and the time taken is several orders of magnitude more than the main-memory reference required in the best case. Hence transaction execution-time variance is very high and can be estimated only poorly. It is therefore difficult to plan schedules which need to finish within a deadline.

24.10 Explain why it may be impractical to require serializability for long-duration transactions.

Answer: In the presence of long-duration transactions, trying to ensure serializability has several drawbacks:-

- **a.** With a waiting scheme for concurrency control, long-duration transactions will force long waiting times. This means that response time will be high, concurrency will be low, so throughput will suffer. The probability of deadlocks is also increased.
- **b.** With a time-stamp based scheme, a lot of work done by a long-running transaction will be wasted if it has to abort.
- **c.** Long duration transactions are usually interactive in nature, and it is very difficult to enforce serializability with interactiveness.

Thus the serializability requirement is impractical. Some other notion of database consistency has to be used in order to support long duration transactions.

24.11 Consider a multithreaded process that delivers messages from a durable queue of persistent messages. Different threads may run concurrently, attempting to deliver different messages. In case of a delivery failure, the message must be restored in the queue. Model the actions that each thread carries out as a multilevel transaction, so that locks on the queue need not be held till a message is delivered.

Answer: Each thread can be modeled as a transaction T which takes a message from the queue and delivers it. We can write transaction T as a multilevel transaction with subtransactions T_1 and T_2. Subtransaction T_1 removes a message from the queue and subtransaction T_2 delivers it. Each subtransaction releases locks once it completes, allowing other transactions to access the queue. If transaction T_2 fails to deliver the message, transaction T_1 will be undone by invoking a compensating transaction which will restore the message to the queue.

24.12 Discuss the modifications that need to be made in each of the recovery schemes covered in Chapter 17 if we allow nested transactions. Also, explain any differences that result if we allow multilevel transactions.

Answer:

- The advanced recovery algorithm of Section 17.9 :-

  The redo pass, which repeats history, is the same as before. We discuss below how the undo pass is handled.

  Recovery with nested transactions:

  Each subtransaction needs to have a unique TID, because a failed subtransaction might have to be independently rolled back and restarted.

  If a subtransaction fails, the recovery actions depend on whether the unfinished upper-level transaction should be aborted or continued. If it should be aborted, all finished and unfinished subtransactions are undone by a backward scan of the log (this is possible because the locks on the modified data items are not released as soon as a subtransaction finishes). If the nested transaction is going to be continued, just

the failed transaction is undone, and then the upper-level transaction continues.

In the case of a system failure, depending on the application, the entire nested-transaction may need to be aborted, or, (for e.g., in the case of long duration transactions) incomplete subtransactions aborted, and the nested transaction resumed. If the nested-transaction must be aborted, the rollback can be done in the usual manner by the recovery algorithm, during the undo pass. If the nested-transaction must be restarted, any incomplete subtransactions that need to be rolled back can be rolled back as above. To restart the nested-transaction, state information about the transaction, such as locks held and execution state, must have been noted on the log, and must restored during recovery. Mini-batch transactions (discussed in Section 21.2.7) are an example of nested transactions that must be restarted.

Recovery with multi-level transactions:

In addition to what is done in the previous case, we have to handle the problems caused by exposure of updates performed by committed subtransactions of incomplete upper-level transactions. A committed subtransaction may have released locks that it held, so the compensating transaction has to reacquire the locks. This is straightforward in the case of transaction failure, but is more complicated in the case of system failure.

The problem is, a lower level subtransaction a of a higher level transaction A may have released locks, which have to be reacquired to compensate A during recovery. Unfortunately, there may be some other lower level subtransaction b of a higher level transaction B that started and acquired the locks released by a, before the end of A. Thus undo records for b may precede the operation commit record for A. But if b had not finished at the time of the system failure, it must first be rolled back and its locks released, to allow the compensating transaction of A to reacquire the locks.

This complicates the undo pass; it can no longer be done in one backward scan of the log. Multilevel recovery is described in detail in David Lomet, "MLR: A Recovery Method for Multi-Level Systems", ACM SIGMOD Conf. on the Management of Data 1992, San Diego.

- Recovery in a shadow paging scheme :-

  In a shadow paging based scheme, the implementation will become very complicated if the subtransactions are to be executed concurrently. If they are to execute serially, the current page table is copied to the shadow page table at the end of every subtransaction. The general idea of recovery then is alike to the logging based scheme, except that undoing and redoing become much easier, as in Section 17.5.

24.13 What is the purpose of compensating transactions? Present two examples of their use.

Answer: A compensating transaction is used to perform a semantic undo of

changes made previously by committed transactions. For example, a person might deposit a check in their savings account. Then the database would be updated to reflect the new balance. Since it takes a few days for the check to clear, it might be discovered later that the check bounced, in which case a compensating transaction would be run to subtract the amount of the bounced check from the depositor's account. Another example of when a compensating transaction would be used is in a grading program. If a student's grade on an assignment is to be changed after it is recorded, a compensating program (usually an option of the grading program itself) is run to change the grade and redo averages, etc.

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

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

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

Answer:

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

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

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

In this schedule, T_{G2} starts only after T_{G1} finishes. Within each site, there is local serializability. In site 1, $T_{G2} \rightarrow T_1 \rightarrow T_{G1}$ is a serializability order.

In site 2, $T_{G1} \rightarrow T_2 \rightarrow T_{G2}$ is a serializability order. Yet the global schedule schedule is non-serializable.

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

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

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

Answer:

a. The same system as in the answer to Exercise 24.14 is assumed, except that now both the global transactions are read-only. Consider the schedule given below.

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

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

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