INSTRUCTOR'S MANUAL TO ACCOMPANY

Database System Concepts

Fifth 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 5^{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.

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 addressed 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.

John Corwin and Swathi Yadlapalli did the bulk of the work in preparing the instructors manual for the 5^{th} edition. This manual is derived from the manuals for the earlier editions. The manual for the 4^{th} edition was prepared by Nilesh Dalvi, Sumit Sanghai, Gaurav Bhalotia and Arvind Hulgeri. 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 created on December 19, 2005

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 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

1.5 List four applications which you have used, that most likely used a database system to store persistent data.
Answer:

- Banking: For account information, transfer of funds, banking transactions.
- Universities: For student information, online assignment submissions, course registrations, and grades.
- Airlines: For reservation of tickets, and schedule information.
- Online news sites: For updating new, maintainence of archives.
- Online-trade: For product data, availability and pricing informations, order-tracking facilities, and generating recommendation lists.

1.6 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.

1.7 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.

1.8 List five responsibilities of a database management system. For each responsibility, explain the problems that would arise if the responsibility were not dis-

charged.
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.

1.9 List at least two reasons why database systems support data manipulation using a declarative query language such as SQL, instead of just providing a library of C or C++ functions to carry out data manipulation.
Answer:

a. Declarative languages are easier for programmers to learn and use (and even more so for non-programmers).
b. The programmer does not have to worry about how to write queries to ensure that they will execute efficiently; the choice of an efficient execution technique is left to the database system. The declarative specification makes it easier for the database system to make a proper choice of execution technique.

1.10 Explain what problems are caused by the design of the table in Figure 1.5.
Answer:

- The customer needs to have an account in order for his information to be included in the table. Nulls can be used when there is no account, but null values are rather difficult to handle.

- When an account is to be deleted, the information about the customer is also lost. Ideally, we would like to have the customer information in the database irrespective of whether he/she has an account without resorting to null values.
- If a user has more than one account, the address information is repeated all over again.

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

- 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

CHAPTER 2

Relational Model

This chapter presents the relational model and the relational-algebra formal query language. The relational model is used extensively throughout the text as is the relational algebra

Section 2.4 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.

Exercises

2.4 Describe the differences in meaning between the terms *relation* and *relation schema*.

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.

2.5 Consider the relational database of Figure 2.35, 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.

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

Figure 2.35. Relational database for Exercises 2.1, 2.3 and 2.9.

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. 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. $\Pi_{person\text{-}name}\ (\sigma_{company\text{-}name} = \text{“First Bank Corporation”}\ (works))$
b. $\Pi_{person\text{-}name,\ city}\ (employee \bowtie (\sigma_{company\text{-}name} = \text{“First Bank Corporation”}\ (works)))$
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. Note: Small Bank Corporation will be included in each answer.
$\Pi_{company\text{-}name}\ (company \div (\Pi_{city}\ (\sigma_{company\text{-}name} = \text{“Small Bank Corporation”}\ (company))))$

2.6 Consider the relation of Figure 2.20, 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. $\Pi_{customer\text{-}name,customer\text{-}city,amount}((borrower \bowtie loan) \supset\!\bowtie customer)$

2.7 Consider the relational database of Figure 2.35. Give an expression in the relational algebra for each request:

a. Give all employees of First Bank Corporation a 10 percent salary raise.
b. 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.
c. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

Answer:

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

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

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

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

$works \leftarrow (works - t_1) \cup t_2$
c. $works \leftarrow works - \sigma_{company-name=\text{“Small Bank Corporation”}}(works)$

2.8 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. $t_1 \leftarrow {}_{account\text{-}number}\mathcal{G}_{\textbf{count}\ customer\text{-}name}(depositor)$
$\Pi_{account\text{-}number}\left(\sigma_{num\text{-}holders>2}\left(\rho_{account\text{-}holders(account\text{-}number,num\text{-}holders)}(t_1)\right)\right)$
b. $t_1 \leftarrow (\rho_{d1}(depositor) \times \rho_{d2}(depositor) \times \rho_{d3}(depositor))$
$t_2 \leftarrow \sigma_{(d1.account\text{-}number=d2.account\text{-}number=d3.account\text{-}number)}(t_1)$

$$\Pi_{d1.account\text{-}number}(\sigma_{(d1.customer\text{-}name \neq d2.customer\text{-}name \wedge d2.customer\text{-}name \neq d3.customer\text{-}name \wedge d3.customer\text{-}name \neq d1.customer\text{-}name)}(t_2))$$

2.9 Consider the relational database of Figure 2.35. 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. $t_1 \leftarrow {}_{company\text{-}name}\mathcal{G}_{\textbf{count-distinct}\ person\text{-}name}(works)$
$t_2 \leftarrow \mathbf{max}_{num\text{-}employees}(\rho_{company\text{-}strength(company\text{-}name,num\text{-}employees)}(t_1))$
$\Pi_{company\text{-}name}(\rho_{t_3(company\text{-}name,num\text{-}employees)}(t_1) \bowtie \rho_{t_4(num\text{-}employees)}(t_2))$

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

c. $t_1 \leftarrow {}_{company\text{-}name}\mathcal{G}_{\textbf{avg}\ salary}(works)$
$t_2 \leftarrow \sigma_{company\text{-}name = \text{“First Bank Corporation”}}(t_1)$
$\Pi_{t_3.company\text{-}name}((\rho_{t_3(company\text{-}name,avg\text{-}salary)}(t_1))$
$\bowtie_{t_3.avg\text{-}salary > first\text{-}bank.avg\text{-}salary} (\rho_{first\text{-}bank(company\text{-}name,avg\text{-}salary)}(t_2)))$

2.10 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.

2.11 Consider the following relational schema

employee(*empno*, *name*, *office*, *age*)
books(*isbn*, *title*, *authors*, *publisher*)
loan(*empno*, *isbn*, *date*)

Write the following queries in relational algebra.

a. Find the names of employees who have borrowed a book published by McGraw-Hill.
b. Find the names of employees who have borrowed all books published by McGraw-Hill.
c. Find the names of employees who have borrowed more than five different books published by McGraw-Hill.
d. For each publisher, find the names of employees who have borrowed more than five books of that publisher.

Answer:

a. $t_1 \leftarrow \Pi_{isbn}(\sigma_{publisher=\text{“}McGraw-Hill''}(books))$
$\Pi_{name}((employee \bowtie loan) \bowtie t_1))$

b. $t_1 \leftarrow \Pi_{isbn}(\sigma_{publisher=\text{“}McGraw-Hill''}(books))$
$\Pi_{name,isbn}(employee \bowtie loan) \div t_1$

c. $t_1 \leftarrow employee \bowtie loan \bowtie (\sigma_{publisher=\text{“}McGraw-Hill''}(books))$
$\Pi_{name}\ (\sigma_{count(isbn) > 5}\ ((\ _{empno}\ G_{count-distinct(isbn)}(t_1))))$

d. $t_1 \leftarrow employee \bowtie loan \bowtie books$
$\Pi_{publisher,name}\ (\sigma_{count(isbn) > 5}\ ((\ _{publisher,empno}\ G_{count-distinct(isbn)}(t_1)))$

CHAPTER 3

SQL

Chapter 3 covers the relational language SQL. The discussion is based on SQL:1999. Integrity constraint and authorization features of 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.

Exercises

3.8 Consider the insurance database of Figure 3.11, where the primary keys are underlined. Construct the following SQL queries for this relational database.

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

b. 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.

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 3.11. Insurance database.

a. SQL query:

```
select   count (distinct *)
from     accident
where    exists
         (select *
          from participated, person, owns
          where participated.driver_id = owns.driver_id
                and participated.driver_id = person.driver_id
                and person.name = 'John Smith'
                and accident.report_number = participated.report_number)
```

b. SQL query:

```
update participated
set damage_amount = 3000
where report_number = "AR2197" and driver_id in
      (select driver_id
       from owns
       where license = "AABB2000")
```

3.9 Consider the employee database of Figure 3.12, 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 all employees in the database who live in the same cities as the companies for which they work.
c. Find all employees in the database who live in the same cities and on the same streets as do their managers.
d. Find all employees who earn more than the average salary of all employees of their company.
e. Find the company that has the smallest payroll.

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'
```

employee (*employee_name*, *street*, *city*)
works (*employee_name*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*employee_name*, *manager_name*)

Figure 3.12. Employee database.

b. 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
```

c. 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
```

d. 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.

```
select employee_name
from works T
where salary > (select avg (salary)
                from works S
                where T.company_name = S.company_name)
```

e. 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)
```

3.10 Consider the relational database of Figure 3.12. Give an expression in SQL for each of the following queries.

a. Give all employees of First Bank Corporation a 10 percent raise.
b. Give all managers of First Bank Corporation a 10 percent raise.
c. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

Answer:

a. Give all employees of First Bank Corporation a 10-percent raise. (the solution assumes that each person works for at most one company.)

```
update works
set salary = salary * 1.1
where company_name = 'First Bank Corporation'
```

b. 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'
```

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

```
delete works
where company_name = 'Small Bank Corporation'
```

3.11 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 $C\ =\ D$

3.12 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 $r1.A$, $r2.B$, $r2.C$
from $r1$, $r2$
where $r1.B\ =\ r2.B$

3.13 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.

3.14 Consider the relational database of Figure 3.12. 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.

3.15 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)
    )
```

3.16 List two reasons why null values might be introduced into the database.
Answer:

- "null" signifies an unknown value.
- "null" is also used when a value does not exist.

3.17 Show how to express the **coalesce** operation from Exercise 3.4 using the **case** operation.
Answer:

```
select
    case Result
        when (A1 is not null) then A1
        when (A2 in not null) then A2
        .
        .
        .
        when (An is not nulll) then An
        else null
    end
from A
```

3.18 Give an SQL schema definition for the employee database of Figure 3.12. 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*))

3.19 Using the relations of our sample bank database, write SQL expressions 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. A view containing the account numbers and customer names (but not the balances) for all accounts at the Deer Park branch.

```
create view custacctinfo as
        select d.account_number, d.customer_name
        from depositor d, account a
        where d.account_number = a.account_number
                and a.branch_name = 'Deer Park'
```

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

```
create view custinfo as
        select c.customer_name, c.customer_street,
        c.customer_city
        from customer c, depositor d
        where c.customerhname not in (select
        customer_name
                from borrower)
        and c.customerhname = d.customer_name
```

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

create view *custavgbal* **as**
select *customer_name*, **avg**(*balance*)
from *account a*, *depositor d*
where *a.account_number* = *d.account_number*
and *branch_name* = 'Rock Ridge'
group by *customer_name*

3.20 For each of the views that you defined in Exercise 3.19, explain how updates would be performed (if they should be allowed at all).
Answer:

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

Updates can be allowed on thie view. If a tuple ("Number", "Name") is inserted to the view, it is equivalent to inserts into 3 tables:

insert into *depositor* ('Name', 'Number')
insert into *account* ('Number', 'Deer Park', **null**)
insert into *customer* ('Name', **null**, **null**)

No primary key or foriegn key constraints are violated, so this update is valid.

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

This update cannot be allowed. When a new tuple is added to the view, it means that a new customer who has an account but no loan is added. The changes should be reflected in 3 tables - customer, account and depositor. There is no information about the account_number, so the account and depositor's primary key constraints are violated. So, this update is not valid.

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

Updates should not be allowed because there is no way to determine how to change the underlying data. As aggregates are involved, there is no definite way to change the data in the base tables. All the values could be changed to the given value, so that the average is the same or the values could be adjusted such that the average of them is equal to the given value. Updates on views involving aggregates are not valid.

3.21 Consider the following relational schema

employee(*empno*, *name*, *office*, *age*)
books(*isbn*, *title*, *authors*, *publisher*)
loan(*empno*, *isbn*, *date*)

Write the following queries in SQL.

a. Print the names of employees who have borrowed any book published by McGraw-Hill.

b. Print the names of employees who have borrowed all books published by McGraw-Hill.

c. For each publisher, print the names of employees who have borrowed more than five books of that publisher.

Answer:

a. Print the names of employees who have borrowed any book published by McGraw-Hill.

```
select name
from employee e, books b, loan l
where e.empno = l.empno
      and l.isbn = b.isbn and
          b.publisher = 'McGrawHill'
```

b. Print the names of employees who have borrowed all books published by McGraw-Hill.

```
select distinct e.name
from employee e
where not exists
      ((select isbn
      from books
      where publisher = 'McGrawHill')
      except
      (select isbn
      from loan l
      where l.empno = e.empno))
```

c. For each publisher, print the names of employees who have borrowed more than five books of that publisher.

```
select publisher, name
from (select publisher, name, count isbn
      from employee e, books b, loan l
      where e.empno = l.empno
      and l.isbn = b.isbn
      group by publisher, name) as
      emppub(publisher, name, count_books)
      where count_books ¡ 5
```

3.22 Consider the relational schema

student(*student_id*, *student_name*)
registered(*student_id*, *course_id*)

Write an SQL query to list the student-id and name of each student along with the total number of courses that the student is registered for. Students who are not registered for any course must also be listed, with the number of registered

courses shown as 0.
Answer:

```
select student_id, student_name, count_courses
from (select student_id, count course_id
      from registered group by student_id) as
      count(student_name, count_courses)), student
      where count.student_id = student.student_id
      union
select student_id, student_name, 0
from student where student_id not in
      (select student_id from registered)
```

3.23 Suppose that we have a relation *marks*(*student_id*, *score*). Write an SQL query to find the *dense rank* of each student. That is, all students with the top mark get a rank of 1, those with the next highest mark get a rank of 2, and so on. Hint: Split the task into parts, using the **with** clause.
Answer:

```
with intermediate(score, rank) as
      (select score, rownum from
      marks order by score)
select m.student_id, rank
from marks m, intermediate i
bf where m.score = i.score
```

CHAPTER 4

Advanced SQL

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.

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.

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.

Exercises

4.7 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).

4.8 Write a Java function using JDBC metadata features that takes a ResultSet as an input parameter, and prints out the result in tabular form, with appropriate names as column headings.
Answer:

```
public class ResultSetTable implements TabelModel {
    ResultSet result;
    ResultSetMetaData metadata;
    int num_cols;

    ResultSetTable(ResultSet result) throws SQLException {
        this.result = result;
        metadata = result.getMetaData();
        num_cols = metadata.getColumnCount();

        for(int i = 1; i <= num_cols; i++) {
            System.out.print(metadata.getColumnName(i) + " ");
        }
        System.out.println();
        while(result.next()) {
            for(int i = 1; i <= num_cols; i++) {
                System.out.print(result.getString(
                    metadata.getColumnName(i) + " "));
            }
            System.out.println();
        }
    }
}
```

4.9 Write a Java function using JDBC metadata features that prints a list of all relations in the database, displaying for each relation the names and types of its

attributes.
Answer:

```
DatabaseMetaData dbmd = conn.getMetaData();
ResultSet rs = dbmd.getTables();
while (rs.next()) {
    System.out.println(rs.getString("TABLE_NAME");
    ResultSet rs1 = dbmd.getColumns(null, "schema-name",
                    rs.getString("TABLE_NAME"), "%");
    while (rs1.next()) {
        System.out.println(rs1.getString("COLUMN_NAME"),
                    rs.getString("TYPE_NAME");
    }
}
```

4.10 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 functions as appropriate.
b. Without using SQL 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")
```

4.11 Rewrite the query in Section 4.6.1 that returns the name, street and city of all customers with more than one account, using the **with** clause instead of using a function call.
Answer:

```
with account_count (customer_name, number) as
        (select customer_name, count (account_number)
             from depositor
             group by customer_name)
select customer_name, customer_street, customer_city
        from customer, account_count
        where customer.customer_name = account_count.customer_name
        and number > 1
```

4.12 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.

4.13 Modify the recursive query in Figure 4.14 to define a relation

empl_depth(*employee_name*, *manager_name*, *depth*)

where the attribute *depth* indicates how many levels of intermediate managers are there between the employee and the manager. Employees who are directly under a manager would have a depth of 0.
Answer:

```
with recursive empl(employee_name, manager_name, depth) as
     (select employee_name, manager_name, 0
          from manager
     union
     select manager.employee_name, empl.manager_name, (empl.depth + 1)
          from manager, empl
          where manager.manager_name = empl.employee_name)

     select *
          from empl
```

4.14 Consider the relational schema

part(*part_id*, *name*, *cost*)
subpart(*part_id*, *subpart_id*, *count*)

A tuple $(p_1, p_2, 3)$ in the *subpart* relation denotes that the part with part-id p_2 is a direct subpart of the part with part-id p_1, and p_1 has 3 copies of p_2 Note that p_2 may itself have further subparts. Write a recursive SQL query that outputs the names of all subparts of the part with part-id "P-100".
Answer:

```
with recursive total-part(name) as
    (select part.name
        from subpart, part
        where subpart.part_id = "P-100" and
        subpart.part_id = part.part_id
    union
    select p2.name
        from subpart s, part p1, part p2
        where s.part_id = p1.part_id
        and p1.name total-part.name
        and s.subpart_id = p2.part_id)

    select *
        from total-part
```

4.15 Consider again the relational schema from Exercise 4.14. Write a JDBC function using non-recursive SQL to find the total cost of part "P-100", including the costs of all its subparts. Be sure to take into account the fact that a part may have multiple occurrences of a subpart. You may use recursion in Java if you wish.
Answer:
The SQL function 'total_cost' is called from within the JDBC code.
SQL function:

```
create function total_cost(id char(10))

returns table(number integer)

begin
    create temporary table result (name char(10), number integer);
    create temporary table newpart (name char(10), number integer);
    create temporary table temp (name char(10), number integer);
    create temporary table final_cost(number integer);

    insert into newpart
        select subpart_id, count
```

```
            from subpart
            where part_id = id
        repeat
            insert into result
            select name, number
            from newpart;

            insert into temp
            (select subpart.subpart_id, count
                from newpart, subpart
                where newpart.subpart_id = subpart.part_id;
            )
            except(
                select subpart_id, count
                    from result;
                );

            delete from newpart;
            insert into newpart
                select *
                from temp;
            delete from temp;

        until not exists(select * from newpart)
        end repeat;

        with part_cost(number) as
            select (count*cost)
                from result, part
                where result.subpart_id = part.part_id);
        insert into final_cost
            select *
            from part_cost;
        return table final_cost;
end
```

JDBC function:

```
Connection conn = DriverManager.getConnection(
    ''jdbc:oracle:thin:@db.yale.edu:2000:bankdb'',
    userid,passwd);
Statement stmt = conn.createStatement();

ResultSet rset = stmt.executeQuery(
    "select SUM(number)
        from table(total_cost("P-100"))");

System.out.println(rset.getFloat(2));
```

CHAPTER 5

Other Relational Languages

In this chapter we study additional relational languages. Two formal langauges — the tuple relational calculus and domain relational calculus and two higher-level languages—QBE and Datalog.

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.

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 See the Tools section at the end of Chapter 5 for the URLs of these systems.

Exercises

5.6 Consider the employee database of Figure 5.14. Give expressions in tuple relational calculus and domain relational calculus 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 that in which 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. $\{t \mid \exists\, s \in works\, (t[person\_name] = s[person\_name] \wedge s[company\_name] = \text{“First Bank Corporation”})\}$
ii. $\{ < p > \mid \exists\, c,\ s\, (< p, c, s > \in works \wedge c = \text{“First Bank Corporation”})\}$

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

i. $\{t \mid \exists\, r \in employee\, \exists\, s \in works\, (\, t[person\_name] = r[person\_name] \wedge t[city] = r[city] \wedge r[person\_name] = s[person\_name] \wedge s[company\_name] = \text{'First Bank Corporation'})\}$
ii. $\{< p, c > \mid \exists\, co,\ sa,\ st\, (< p, co, sa > \in works \wedge < p, st, c > \in employee \wedge co = \text{“First Bank Corporation”})\}$

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. $\{t \mid t \in employee \wedge (\exists\, s \in works\, (\, s[person\_name] = t[person\_name] \wedge s[company\_name] = \text{“First Bank Corporation”} \wedge s[salary] > 10000))\}$

ii. $\{< p, s, c > \mid < p, s, c > \in employee \wedge \exists\, co,\ sa\, (< p, co, sa > \in works \wedge co = \text{'First Bank Corporation'} \wedge sa > 10000)\}$

d. Find the names of all employees in this database who live in the same city as the company for which they work:

i. $\{t \mid \exists\, e \in employee\, \exists\, w \in works\, \exists\, c \in company$
$(t[person\_name] = e[person\_name]$
$\wedge\, e[person\_name] = w[person\_name]$
$\wedge\, w[company\_name] = c[company\_name] \wedge e[city] = c[city])\}$

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

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

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

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. $\{t \mid \exists\, w \in works\, (t[person\_name] = w[person\_name] \wedge \forall\, s \in works$
$(s[company\_name] =$ "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. $\{t \mid \forall\, s \in company\, (s[company\_name] = \text{“Small Bank Corporation”} \Rightarrow \exists\, r \in company\, (t[company\_name] = r[company\_name] \wedge r[city] = s[city]))\}$

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

5.7 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.

5.8 Repeat Exercise 5.7, writing SQL queries instead of relational-algebra expressions.
Answer:

a. **select** a
from r
where $b = 17$

b. **select** a, b, c
from r, s
where $r.a = s.a$

c. (**select** a
from r)
union
(**select** a
from s)

d. **select** a
from r **as** $r1$, r **as** $r2$, s
where $r1.a = s.a$ **and** $r2.a = s.c$ **and** $r1.b > r2.b$

5.9 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. $r \ltimes\!\sqsubset s$
b. $r \supset\!\!\ltimes\!\sqsubset s$
c. $r \supset\!\!\ltimes s$

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

5.10 Consider the insurance database of Figure 5.15, where the primary keys are underlined. Construct the following GQBE 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.

Answer:

a.

participated	driver_id	car	report_number	damage
	_x		_r	

accident	report_number	date	location
	_r	_d	

owns	driver_id	licence	
	_x		P.COUNT._x

conditions
_d = ($\geq$ "1/1/1989" **AND** $<$ "1/1/1990")

b.

person	driver_id	name	address
	_x	John Smith	

participated	driver_id	car	report_number	damage
	_x		_r	

owns	driver_id	licence
	_x	_y

car	licence	model	year
	_y		

accident	report_number	date	location	
	_r			P.COUNT._r

5.11 Give a tuple-relational-calculus expression to find the maximum value in relation $r(A)$.
Answer: $\{a \mid \forall\, b \in R,\ a >= b\}$

5.12 Repeat Exercise 5.6 using QBE and Datalog.
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 Co	

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.

employee	*person_name*	*street*	*city*
	P._x		P._y

works	*person_name*	*company_name*	*salary*
	_x	First Bank Corporation	

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 Corporation	>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*	*salary*
	Small Bank Corporation	_*x*
	P. _*c*	_*y*
	Small Bank Corporation	_*y*

condition
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)*

5.13 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
	_a	_b	_c

r2	A	B	C
	_d	_e	_f

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
	_a	_b	_c

r2	A	B	C
	_d	_e	_f

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		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)*

5.14 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**

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

CHAPTER 6

Database Design and the E-R 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 6.6), design constraints (Section 6.7.4) and aggregation (Section 6.7.5), and the corresponding subsections of Section 6.9 (Reduction of an E-R Schema to Tables) can be omitted if time is short. We recommend covering specialization (Section 6.7.1) at least in some detail, since it is an important concept for object-oriented databases (Chapter 9).

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.

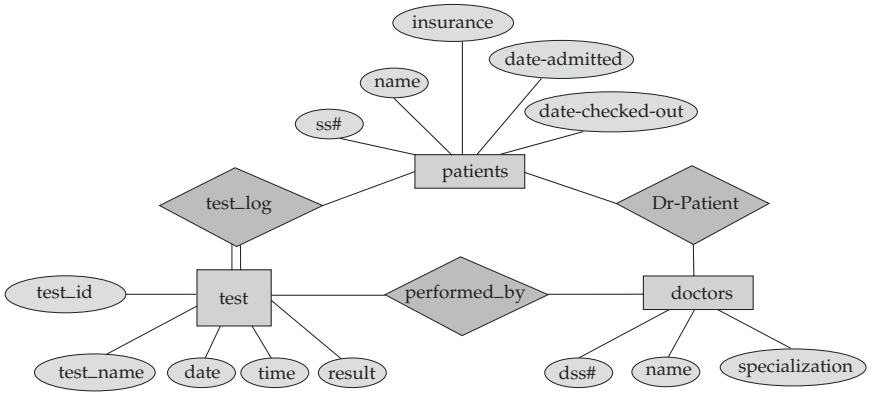


Figure 6.1 E-R diagram for a hospital.

Exercises

6.14 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.

6.15 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 6.1

6.16 Construct appropriate tables for each of the E-R diagrams in Practice Exercises 6.1 and 6.2.

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)

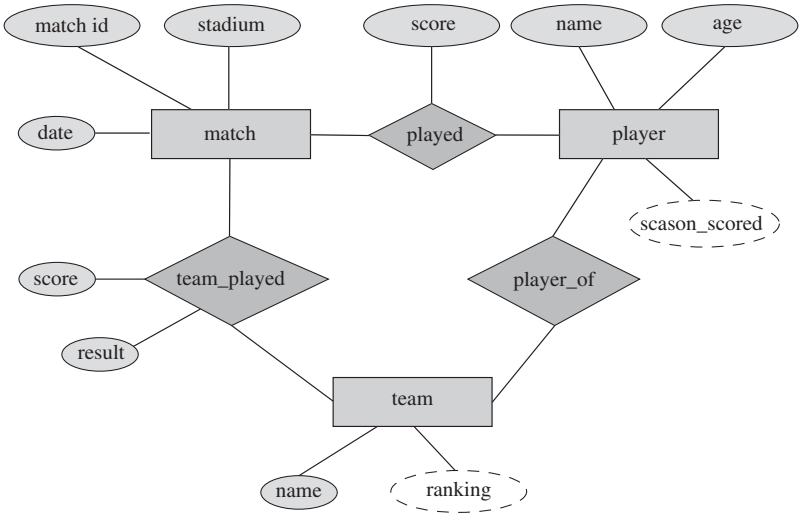


Figure 6.2 E-R diagram for all teams statistics.

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>)

6.17 Extend the E-R diagram of Practice Exercise 6.4 to track the same information for all teams in a league.
Answer: See Figure 6.2 Note that a player can stay in only one team during a season.

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

6.19 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.

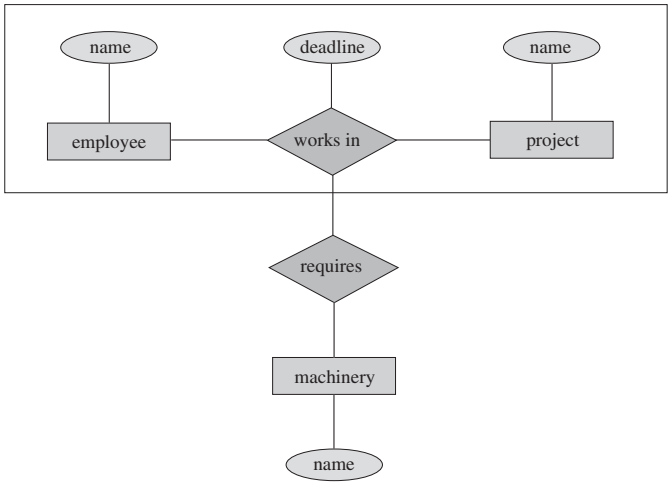


Figure 6.3 E-R diagram: Example 1 of aggregation.

- 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.

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

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 6.3

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 6.4

6.21 Consider the E-R diagram in Figure 6.31, 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: Nn answer

6.22 In Section 6.5.3, we represented a ternary relationship (repeated in Figure 6.29a) using binary relationships, as shown in Figure 6.29b. Consider the alternative shown in Figure 6.29c. Discuss the relative merits of these two alternative representations of a ternary relationship by binary relationships.
Answer: The model of Figure 6.29c will not be able to represent all ternary relationships. Consider the *ABC* relationship set below.

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*.

6.23 Consider the relation schemas shown in Section 6.9.7, which were generated from the E-R diagram in Figure 6.25. For each schema, specify what foreign-key constraints, if any, should be created.
Answer: To determine the foreign key constrains, follow the reduction rules specified in Section 6.9.

- In the *dependent_name* relation, the *employee_id* attribute references the *employee* relation.
- In the *account_branch* relation, the *account_number* attribute references the *account* relation and the *branch_name* attribute references the *branch* relation.
- In the *loan_branch* relation, the *loan_number* attribute references the *loan* relation and the *branch_name* attribute references the *branch* relation.

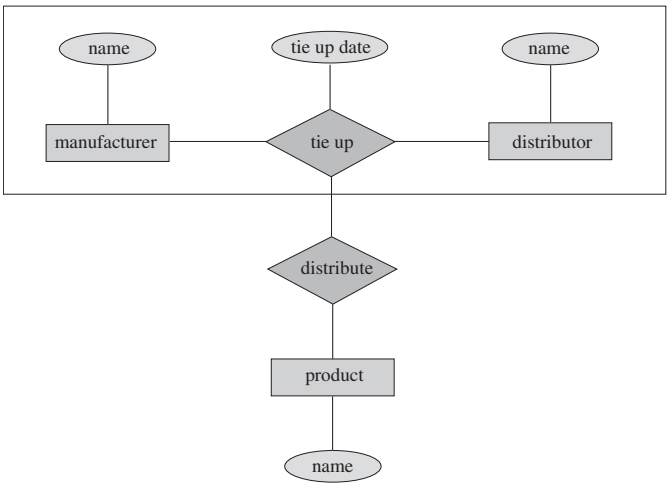


Figure 6.4 E-R diagram: Example 2 of aggregation.

- In the *borrower* relation, the *customer_id* attribute references the *customer* relation and the *loan_number* attribute references the *loan* relation.
- In the *depositor* relation, the *cutomer_id* attribute references the *customer* relation and the *account_number* attribute references the *account* relation.
- In the *cust_banker* relation, the *cutomer_id* attribute references the *customer* relation and the *employee_id* attribute references the *employee* relation.
- In the *works_for* relation, both the *worker_employee_id* and the *manager_employee_id* attributes reference the *employee* relation.
- In the *payment* relation, the *loan_number* attribute references the *loan* relation.
- In the *savings_account* relation, the *account_number* attribute references the *account* relation.
- In the *checking_account* relation, the *account_number* attribute references the *account* relation.

6.24 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 6.5 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 vehicle 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*.

6.25 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.

6.26 Explain the distinction between disjoint and overlapping constraints.

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

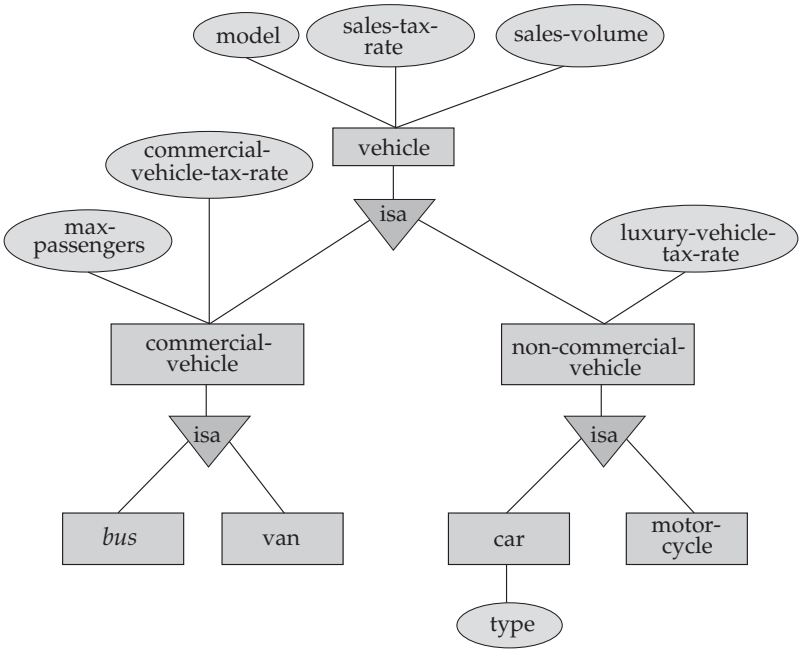


Figure 6.5 E-R diagram of motor-vehicle sales company.

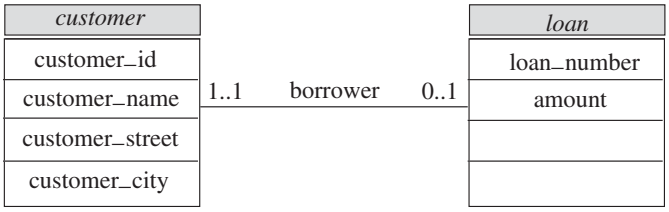


Figure 6.6 UML equivalent of Figure 6.8c.

employee-workteam example of the book, a manager may participate in more than one work-team.

6.27 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.

6.28 Draw the UML equivalents of the E-R diagrams of text Figures 6.8c, 6.9, 6.11, 6.12 and 6.20.
Answer: See Figures 6.6 to 6.10

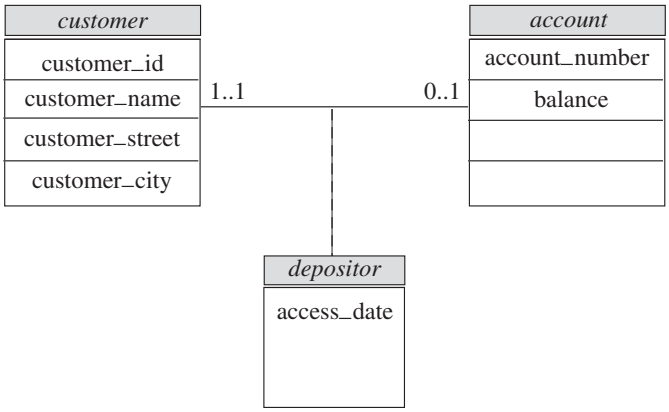


Figure 6.7 UML equivalent of Figure 6.9

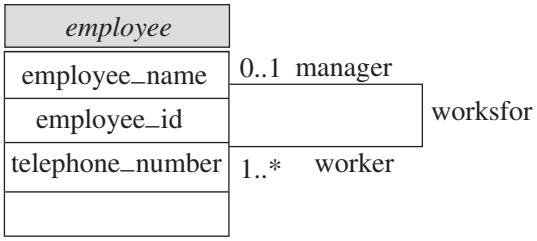


Figure 6.8 UML equivalent of Figure 6.11

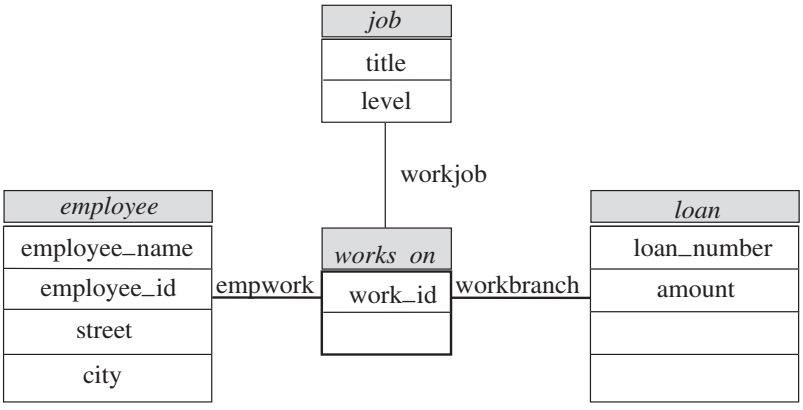


Figure 6.9 UML equivalent of Figure 6.12

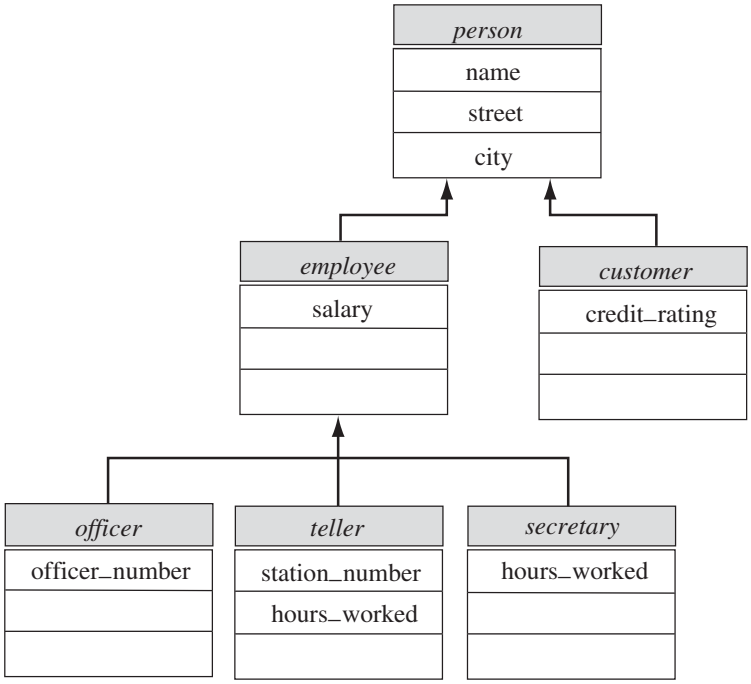


Figure 6.10 UML equivalent of Figure 6.20

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 and 7.3 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.

Exercises

7.17 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.

7.18 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.

7.19 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

7.20 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.

7.21 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

7.22 Using the functional dependencies of Practice Exercise 7.6, compute B^+.
Answer: Computing B^+ by the algorithm in Figure 7.9 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 $B^+ = \{B, D\}$

7.23 Show that the following decomposition of the schema *R* of Practice Exercise 7.1 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.

7.24 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.

7.25 Give a lossless-join decomposition into BCNF of schema R of Exercise 7.1.
Answer: From Exercise 7.6, we know that $B \rightarrow D$ is nontrivial and the left hand side is not a superkey. By the algorithm of Figure 7.12 we derive the relations $\{(A, B, C, E), (B, D)\}$. This is in BCNF.

7.26 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.

7.27 Give a lossless-join, dependency-preserving decomposition into 3NF of schema R of Practice Exercise 7.1.
Answer: First we note that the dependencies given in Practice Exercise 7.1 form a canonical cover. Generating the schema from the algorithm of Figure 7.13 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.

7.28 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 Practice Exercise 7.15 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.

7.29 Given a relational schema $r(A, B, C, D)$, does $A \twoheadrightarrow BC$ logically imply $A \twoheadrightarrow B$ and $A \twoheadrightarrow C$? If yes prove it, else give a counter example.
Answer: $A \twoheadrightarrow BC$ holds on the following table:

r :

A	B	C	D
a_1	b_1	c_1	d_1
a_1	b_2	c_2	d_2
a_1	b_1	c_1	d_2
a_1	b_2	c_2	d_1

If $A \twoheadrightarrow B$, then we know that there exists t_1 and t_3 such that $t_1[B] = t_3[B]$. Thus, we must choose one of the following for t_1 and t_3:

- $t_1 = r_1$ and $t_3 = r_3$, or $t_1 = r_3$ and $t_3 = r_1$:
  Chosing either $t_2 = r_2$ or $t_2 = r_4$, $t_3[C] \neq t_2[C]$.
- $t_1 = r_2$ and $t_3 = r_4$, or $t_1 = r_4$ and $t_3 = r_2$:
  Chosing either $t_2 = r_1$ or $t_2 = r_3$, $t_3[C] \neq t_2[C]$.

Therefore, the condition $t_3[C] = t_2[C]$ can not be satisfied, so the conjecture is false.

7.30 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 Practice Exercise 7.16), we observe that decomposition into 4NF does not lose information provided that a lossless join decomposition is used, yet redundancy is reduced.

CHAPTER 8

Application Design Development

Common Code and Files used in the Solutions

File *jdbc.properties*

```
driver=oracle.jdbc.driver.OracleDriver
url=jdbc:oracle:thin:@localhost:1521:orcl
user=scott
password=tiger
```

A common method for creating new JDBC connection

```
private Connection getConnection() throws Exception {
    FileInputStream fis = new FileInputStream("jdbc.properties");
    Properties props = new Properties();
    props.load(fis);
    String driver = props.getProperty("driver");
    String url= props.getProperty("url");
    String user = props.getProperty("user");
    String passwd = props.getProperty("password");
    Class.forName(driver);
    return DriverManager.getConnection(url, user, passwd);
}
```

Exercises

8.8 Write a servlet and associated HTML code for the following very simple application: A user is allowed to submit a form containing a value, say n, and should get a response containing n "*" symbols.

Answer:
HTML form

```
<html>
    <head>
        <title>DB Book Exercise 8.8 </title>
    </head>
    <form action="servlet/StarServlet" method=get>
        Enter the value for "n"
        <br>
        <input type=text size=5 name="n">
        <input type=submit value="submit">
    </form>
</html>
```

Servlet Code

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class StarServlet extends HttpServlet {

    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        int n = Integer.parseInt(request.getParameter("n"));
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE>Exercise 8.8</TITLE></HEAD>");
        out.println("<BODY>");
        for (int i = 0; i < n; i++) {
            out.print("*");
        }
        out.println("</BODY>");
        out.close();
    }
}
```

8.9 Write a servlet and associated HTML code for the following simple application: A user is allowed to submit a form containing a number, say n, and should get a response saying how many times the value n has been submitted previously. The number of times each value has been submitted previously should be stored in a database.

Answer:
HTML form

```
<html>
    <head>
        <title>DB Book Exercise 8.9 </title>
    </head>
    <form action="servlet/KeepCountServlet" method=get>
        Enter the value for "n"
        <br>
        <input type=text size=5 name="n">
        <input type=submit value="submit">
    </form>
</html>
```

Schema

```
CREATE TABLE SUBMISSION_COUNT (value integer unique, scount integer not null);
```

Servlet Code

```
import java.io.*;
import java.sql.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class KeepCountServlet extends HttpServlet {
    private static final String query =
        "SELECT scount FROM SUBMISSION_COUNT WHERE value=?";

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException
    {
        int n = Integer.parseInt(request.getParameter("n"));
        int count = 0;
        try {
            Connection conn = getConnection();
            PreparedStatement pstmt = conn.prepareStatement(query);
            pstmt.setInt(1, n);
            ResultSet rs = pstmt.executeQuery();
            if (rs.next()) {
                count = rs.getInt(1);
            }
            pstmt.close();

            Statement stmt = conn.createStatement();
            if (count == 0) {
                stmt.executeUpdate("INSERT INTO SUBMISSION_COUNT VALUES (" +
                        n + ", 1)");
            }
            else {
                stmt.executeUpdate("UPDATE SUBMISSION_COUNT SET " +
                        "scount=scount+1 WHERE value=" + n);
            }
            stmt.close();
            conn.close();
        }
        catch(Exception e) {
            throw new ServletException(e.getMessage());
        }
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE>Exercise 8.9</TITLE></HEAD>");
        out.println("<BODY>");
        out.println("The value " + n + " has been submitted " + count + " times previously.");
        out.println("</BODY>");
        out.close();
    }
}
```

8.10 Write a servlet that authenticates a user (based on user names and passwords stored in a database relation), and sets a session variable called *userid* after authentication.

Answer:

HTML form

```
<html>
    <head>
        <title>DB Book Exercise 8.10 </title>
    </head>
    <form action="servlet/SimpleAuthServlet" method=get>
        User Name:
        <input type=text size=20 name="user">
        <BR>
        <BR>
        Password :
        <input type=password size=20 name="passwd">
        <BR>
        <input type=submit value="submit">
    </form>
</html>
```

Schema

```
CREATE TABLE USERAUTH(userid integer primary key, username varchar(100) unique,
                      password varchar(20));
```

Servlet Code

```
import java.io.*;
import java.sql.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleAuthServlet extends HttpServlet {
    private static final String query =
            "SELECT userid, password FROM USERAUTH WHERE username=?";

    public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException
    {
        String user = request.getParameter("user");
        String passwd = request.getParameter("passwd");

        String dbPass = null;
        int userId = -1;
        try {
            Connection conn = getConnection();
            PreparedStatement pstmt = conn.prepareStatement(query);
            pstmt.setString(1, user);
            ResultSet rs = pstmt.executeQuery();


            if (rs.next()) {
                userId = rs.getInt(1);
                dbPass = rs.getString(2);
            }
            pstmt.close();
            conn.close();
        }
        catch(Exception e) {
            throw new ServletException(e.getMessage());
        }
        String message;

        if(passwd.equals(dbPass)) {
            message = "Authentication successful";
            getServletContext().setAttribute("userid", new Integer(userId));
        }
        else {
            message = "Authentication failed! Please check the username and password.";
        }

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<HEAD><TITLE>Exercise 8.10</TITLE></HEAD>");
        out.println("<BODY>");
        out.println(message);
        out.println("</BODY>");
        out.close();
    }
}
```

8.11 What is an SQL injection attack? Explain how it works, and what precautions must be taken to prevent SQL injection attacks.
Answer: SQL injection attack occurs when a malicious user (attacker) manages to get an application to execute an SQL query created by the attacker. If an application constructs an SQL query string by concatenating the user supplied parameters, the application is prone to SQL injection attacks.

For example, suppose an application constructs and executes a query to retrieve a user's password in the following way:

```
String userid = request.getParameter("userid");
executeQuery("SELECT password FROM userinfo WHERE userid= ' " + userid + " ' ");
```

Now, if a user types the value for the parameter as:

```
john' OR userid= 'admin
```

the query constructed will be:

```
SELECT password FROM userinfo WHERE userid='john' OR userid='admin';
```

This can reveal unauthorized information to the attacker.
Prevention:
Use prepared statements.

8.12 Write pseudocode to manage a connection pool. Your pseudocode must include a function to create a pool (providing a database connection string, database user name and password as parameters), a function to request a connection from the pool, a connection to release a connection to the pool, and a function to close the connection pool.
Answer:

```
// This connection pool manager is NOT thread-safe.

INITIAL_POOL_SIZE = 20;
POOL_SIZE_INCREMENT = 5;
MAX_POOL_SIZE = 100;
Queue freeConnections = empty queue;
Queue activeConnections = empty queue;
String poolConnURL;
String poolConnUserId;
String poolConnPasswd;

createPool(connString, userid, password) {
    poolConnURL = connString;
    poolConnUserId = userid;
    poolConnPasswd = password;
    for (i = 0; i < INITIAL_POOL_SIZE; i++) {
        conn = createConnection(connString, userid, password);
        freeConnections.add(conn);
    }
}

Connection getConnection() {
    if(freeConnections.size() != 0){
        conn = freeConnections.remove();
        activeConnections.add(conn);
        return conn;
    }
    activeConns = activeConnections.size();
    if (activeConns == MAX_POOL_SIZE)
        ERROR("Max pool size reached");
    if (MAX_POOL_SIZE - activeConns > POOL_SIZE_INCREMENT)
        connsToCreate = POOL_SIZE_INCREMENT;
    else
        connsToCreate = MAX_POOL_SIZE - activeConns;

    for (i = 0; i < connsToCreate; i++) {
        conn = createConnection(poolConnURL, poolConnUserId, poolConnPasswd);
        freeConnections.add(conn);
    }
    return getConnection();
}

releaseConnection(conn) {
    activeConnections.remove(conn);
    freeConnections.add(conn);
}
```

```
closePool() {
        if(activeConnections.size() != 0)
                WARNING("Connections active. Will force close.");
        for (i=0; i < freeConnections.size(); i++) {
                conn = freeConnections.elementAt(i);
                freeConnections.removeElementAt(i);
                conn.close();
        }

        for (i=0; i < activeConnections.size(); i++) {
                conn = activeConnections.elementAt(i);
                activeConnections.removeElementAt(i);
                conn.close();
        }
}
```

8.13 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.
For the above example, we create a **before delete** trigger on relation s. The trigger first deletes all the tuples of relation r that refer to the tuples being deleted from s.

```
create trigger cascade_delete before delete on s
referencing old row as oldrow
for each row
begin
        delete from r where r.B=oldrow.A;
end
```

8.14 The execution of a trigger can cause another action to be triggered. Most database systems place a limit on how deep the nesting can be. Explain why they might place such a limit.
Answer: It is possible that a trigger body is written in such a way that a nonterminating recursion may result. An example of such a trigger is a *before insert* trigged on a relation that tries to insert another record into the same relation.

In general, it is extremely difficult to statically identify and prohibit such triggers from being created. Hence database systems, at runtime, put a limit on the depth of nested trigger calls.

8.15 Explain why, when a manager, say Mary, grants an authorization, the grant should be done by the manager role, rather than by the user Mary.
Answer: If the grant is done by the user Mary, and at a later time, if the *manager* role is revoked from Mary with *cascading* option then all the users and roles who received a privilege from Mary by virtue of her being a manager, will loose those privileges. This may not be a desirable effect. For example, assume Mary, while having the role *manager*, has granted the role *teller* to John. Now, if the *manager* role gets revoked from Mary (perhaps because Mary leaves the company) with *cascade* option, then John will loose the *teller* role even though John continues to be an employee of the company. To avoid such a problem it is preferable to have the *grant* to be done by the role and not the individual user.

8.16 Suppose user A, who has all authorizations on a relation r, grants select on relation r to **public** with grant option. Suppose user B then grants select on r to A. Does this cause a cycle in the authorization graph? Explain why.
Answer: Yes, it does cause the following cycle in the authorization graph.

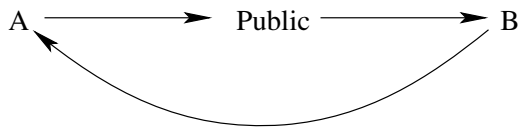


Figure 8.1 Cycle in authorization graph

For each privilege granted to *public*, an edge must be placed between *public* and all users in the system. If this is not done, then the user will not have a path from the root (DBA).

8.17 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: Some security concerns for a bank:

- The disk or backup media containing customer data could be stolen or copied (physical security).
- Authorized bank employees could reveal sensitive customer information to outsiders (human security).
- Files containing database data or logs could be copied (OS security)
- Unauthorized bank employee accesses/updates the bank database (Database security)

8.18 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:
Advantages:

- No need to implement the security and authorization inside the DBMS. This may result in reduced cost.
- Administrators need not learn new commands or use of a new UI. They can create and administer user accounts on the OS they may already be familiar with.
- No worry of unauthorized database users having direct access to the OS files and thus bypassing the database security.

Disadvantages:

- Database users must correspond to operating system users
- Most operating systems do not distinguish between insert, update and delete, they just have a coarse level privilege called "modify"
- Granting/revoking privileges on specific columns of a relation won't be possible.
- Privileges such as "references" cannot be provided

8.19 Oracle's VPD mechanism implements row-level security by adding predicates to the where clause of each query. Give an example of a predicate that could be used to implement row-level security, and three queries with the following properties:

a. For the first query, the query with the added predicate gives the same result as the same as the original query.

b. For the second query, the query with the added predicates gives a result that is always a subset of the original query result.

c. For the third query, the query with the added predicate give incorrect answers.

Answer: A possible predicate to implement row-level security on student table could be:

```
student.name=syscontext.user_name()
```

The following answers assume that the database userid matches the student name.

a. A query to retrieve the details of the logged in student:

```
select * from student where name=<logged in user name>;
```

b. A query to retrieve the details of all the students:

```
select * from student;
```

c. A query to find the average age of the students.

```
select avg(age) from student;
```

8.20 What are two advantages of encrypting data stored in the database?
Answer:

a. Unauthorized users who gain access to the OS files in which the DBMS stores the data cannot read the data.

b. If the application encrypts the data before it reaches the database, it is possible to ensure privacy for the user's data such that even privileged users like database administrators cannot access other users' data.

8.21 Suppose you wish to create an audit trail of changes to the *account* relation.

a. Define triggers to create an audit trail, logging the information into a relation called, for example, *account_trail*. The logged information should include the user-id (assume a function *user_id*() provides this information) and a time stamp, in addition to old and new values. You must also provide the schema of the *account_trail* relation.

b. Can the above implementation guarantee that updates made by a malicious database administrator (or someone who manages to get the administrator's password) will be in the audit trail? Explain your answer.

Answer: Part-a
Schema for the account_trail table

```
account_trail   (user_id integer, timestamp datetime, operation_code integer,
                 new_account_number, new_branch_name, new_balance,
                 old_account_number, old_branch_name, old_balance)
```

Trigger for INSERT

```
create trigger account_insert after insert on account
referencing new row as nrow
for each row
begin
    insert into account_trail values (user_id(), systime(), 1,
    nrow.account_number, nrow.branch_name, nrow.balance, null, null, null);
end
```

Trigger for UPDATE

```
create trigger account_update after update on account
referencing old row as orow, referencing new row as nrow
for each row
begin
    insert into account_trail values (user_id(), systime(), 2, nrow.account_number,
                nrow.branch_name, nrow.balance, orow.account_number,
                orow.branch_name, orow.balance);
end
```

Trigger for DELETE

```
create trigger account_delete after delete on account
referencing old row as orow
for each row
begin
    insert into account_trail values (user_id(), systime(), 3, null, null, null,
            orow.account_number, orow.branch_name, orow.balance);
end
```

Part-b No. Someone who has the administrator privileges can disable the trigger and thus bypass the trigger based audit trail.

8.22 Hackers may be able to fool you into believing that their Web site is actually a Web site (such as a bank or credit card Web site) that you trust. This may be done by misleading email, or even by breaking into the network infrastructure and re-routing network traffic destined for, say mybank.com, to the hackers site. If you enter your user name and password on the hackers site, the site can record it, and use it later to break into your account at the real site. When you use a URL such as https://mybank.com, the the HTTPS protocol is used to prevent such attacks. Explain how the protocol might use digital certificates to verify authenticity of the site.

Answer: In the HTTPS protocol, a Web site first sends a digital certificate to the user's browser. The browser decrypts the digital certificate using the stored public key of the trusted certification authority and displays the site's name from the decrypted message. The user can then verify if the site name matches the one he/she intended to visit (in this example mybank.com) and accept the certificate. The browser then uses the site's public key (that is part of the digital certificate) to encrypt user's data. Note that it is possible for a malicious user to gain access to the digital certificate of mybank.com, but since the user's data (such as password) is encrypted using the public key of mybank.com, the malicious user will not be able to decrypt the data.

8.23 Explain what is a challenge-response system for authentication. Why is it more secure than a traditional password-based system?

Answer: In a challenge-response system, a secret password is issued to the user and is also stored on the database system. When a user has to be authenticated, the database system sends a challenge string to the user. The user encrypts the challenge string using his/her secret password and returns the result. The database system can verify the authenticity of the user by decrypting the string with the same secret password and checking the result with the original challenge string.

The challenge-response system is more secure than a traditional password-based system since the password is not transferred over the network during authentication.

CHAPTER 9

Object-Based 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.

Exercises

9.7 Redesign the database of Exercise 9.2 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.

9.8 Consider the schema from Practice Exercise 9.2.

a. Give SQL:2003 DDL statements to create a relation *EmpA* which has the same information as *Emp*, but where multiset valued attributes *ChildrenSet*, *SkillsSet* and *ExamsSet* are replaced by array valued attributes *ChildrenArray*, *SkillsArray* and *ExamsArray*.

b. Write a query to convert data from the schema of *Emp* to that of *EmpA*, with the array of children sorted by birthday, the array of skills by the skill type and the array of exams by the year.

c. Write an SQL statement to update the *Emp* relation by adding a child Jeb, with a birthdate of February 5, 2001, to the employee named George.

d. Write an SQL statement to perform the same update as above but on the *EmpA* relation. Make sure that the array of children remains sorted by year.

Answer:

a.

```
create type Exams (year int, city varchar(30))
create type SkillsA (type varchar(30),
    ExamsArray Exams array [20])
create type Children (name varchar(30), birthday date)
create table EmpA (ename varchar(30),
    ChildrenArray Children array [10],
    SkillArray SkillsA array [25])
```

b.

```
select ename,
    array(select name, birthday
          from unnest(E.ChildrenSet) as CS
          order by CS.birthday) as ChildrenArray,
    array(select type,
         array(select year, city
               from unnest(SS.ExamSet)
               order by SS.year) as ExamsArray
          from unnest(E.SkillSet) as SS)
```

```
        as SkillsArray
from Emp as E
```

c.

```
update Emp
set ChildrenSet = ChildrenSet union
    multiset[(''Jeb'', ''2/5/2001'')]
where ename = ''George''
```

d. We make use of the infix array concatenation operator, "||", which takes two arrays as arguments and returns the concatenation of the two arrays.

```
update EmpA
set ChildrenArray = array(
    select name, birthday
    from unnest(ChildrenArray ||
        array[(''Jeb'', ''2/5/2001'')])
    order by birthday)
where ename = ''George''
```

9.9 Consider the schemas for the table *people*, and the tables *students* and *teachers*, which were created under *people*, in Section 9.4. 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.

9.10 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.

9.11 Give an SQL:1999 schema definition of the E-R diagram in Figure 9.7, which contains specializations, using subtypes and subtables.
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
```

9.12 Suppose a JDO database had an object A, which references object B, which in turn references object C. Assume all objects are on disk initially. Suppose a program first dereferences A, then dereferences B by following the reference from A, and then finally dereferences C. Show the objects that are represented in memory after each dereference, along with their state (hollow or filled, and values in their reference fields).
Answer: See figures 9.1 through 9.3. Gray boxes indicate persistent objects and white boxes indicate hollow objects.

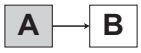


Figure 9.1 State of the program after A is referenced.

Figure 9.2 State of the program after B is referenced.

Figure 9.3 State of the program after C is referenced.

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

10.10 Show, by giving a DTD, how to represent the Non-1NF *books* relation from Section 9.2, 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 )>
] >
```

10.11 Write the following queries in XQuery, assuming the DTD from Practice Exercise 10.2.

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
```

10.12 Consider the XML data shown in Figure 10.2. Suppose we wish to find purchase orders that ordered two or more copies of the part with identifier 123. Consider the following attempt to solve this problem:

```
for $p in purchaseorder
where $p/part/id = 123 and $p/part/quantity >= 2
return $p
```

Explain why the query may return some purchase orders that order less than two copies of part 123. Give a correct version of the above query.
Answer: Reason:
The expression $x =$y evaluates to true if any of the values returned by the first expression is equal to any one of the values returned by the second expression. In this case, if any of the values in the part/quantity $>= 2$ then it evaluates to true, so the query returns parts which have id = 123 irrespective of the quantity.

Query:

```
for $b in purchaseorder
where some $p in $b/part satisfies
      $p/id = 123 AND $p/quantity >= 2
return {$b}
```

10.13 Give a query in XQuery to flip the nesting of data from Exercise 10.10. 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:

```
<books>
      for $x in /books
            $y in /books[author = $x/author]
      return {$y/author $y/title $y/publisher $y/keyword}
</books>
```

10.14 Give the DTD for an XML representation of the information in Figure 6.31. 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.

10.15 Give an XMLSchema representation of the DTD from Exercise 10.14.
Answer:

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="basket" type="BasketType">
  <xs:element name="customer">
    <xs:complexType>
    <xs:attribute name="email" use="required">
    <xs:sequence>
      <xs:element name="name" type="xs:string">
      <xs:element name="address" type="xs:string">
```

```
<!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

```
      <xs:element name="phone" type="xs:decimal">
    </xs:sequence>
      </xs:complexType>
</xs:element>
<xs:key name="customerKey">
  <xs:selector xpath="/bookstore/customer">
  <xs:field xpath="email">
</xs:key>
<xs:element name="book">
  <xs:complexType>
    <xs:attribute name="ISBN" use="required">
    <xs:sequence>
    <xs:element name="year" type="xs:decimal">
    <xs:element name="title" type="xs:string">
    <xs:element name="price" type="xs:decimal">
    <xs:element name="written-by" type="xs:string">
    <xs:element name="published-by" type="xs:string">
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:key name="bookKey">
  <xs:selector xpath="/bookstore/book">
  <xs:field xpath="ISBN">
</xs:key>
<xs:element name="warehouse">
  <xs:complexType>
    <xs:attribute name="code" use="required">
    <xs:sequence>
    <xs:element name="address" type="xs:string">
    <xs:element name="phone" type="xs:decimal">
    <xs:element name="stocks" type="xs:decimal">
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:key name="warehouseKey">
  <xs:selector xpath="/bookstore/warehouse">
  <xs:field xpath="code">
</xs:key>
<xs:element name="author">
  <xs:complexType>
    <xs:attribute name="authid" use="required">
    <xs:sequence>
    <xs:element name="name" type="xs:string">
    <xs:element name="address" type="xs:string">
    <xs:element name="URL" type="xs:string">
    </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>
  <xs:key name="authorKey">
    <xs:selector xpath="/bookstore/author">
    <xs:field xpath="authid">
  </xs:key>
  <xs:element name="publisher">
    <xs:complexType>
      <xs:attribute name="name" use="required">
      <xs:sequence>
      <xs:element name="address" type="xs:string">
      <xs:element name="phone" type="xs:decimal">
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:key name="publisherKey">
    <xs:selector xpath="/bookstore/publisher">
    <xs:field xpath="name">
  </xs:key>
  <xs:element name="basket-of">
    <xs:attribute name="owner" use="required">
  </xs:element>
  <xs:keyref name="basketCustomerFKey" refer="customerKey">
    <xs:selector xpath="/bookstore/customer">
    <xs:field xpath="owner">
  </xs:key>
  <xs:element name="contains">
    <xs:attribute name="book" use="required">
    <xs:attribute name="number" use="required" type="xs:decimal"'>
  </xs:element>
  <xs:keyref name="bookBasketFKey" refer="bookKey">
    <xs:selector xpath="/bookstore/book">
    <xs:field xpath="book/">
  </xs:key>
  <xs:element name="stocks">
    <xs:attribute name="book" use="required">
    <xs:attribute name="number" use="required" type="xs:decimal"'>
  </xs:element>
  <xs:keyref name="bookwarehouseFKey" refer="bookKey">
    <xs:selector xpath="/bookstore/book">
    <xs:field xpath="book/">
  </xs:key>
  <xs:element name="written-by">
    <xs:attribute name="authors" use="required">
  </xs:element>
  <xs:keyref name="authorbookFKey" refer="authorKey">
```

```
<!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 10.15. DTD for bibliographical data.

```
        <xs:selector xpath="/bookstore/author">
        <xs:field xpath="authors">
      </xs:key>
      <xs:element name="published-by">
        <xs:attribute name="publisher" use="required">
      </xs:element>
      <xs:keyref name="bookpublisherFKey" refer="publisherKey">
        <xs:selector xpath="/bookstore/publisher">
        <xs:field xpath="publisher">
      </xs:key>
        <xs:complexType name="BasketType">
          <xs:attribute name="basketid" use="required">
          <xs:sequence>
          <xs:element name="contains" minOccurs="1" maxOccurs="unbounded">
          <xs:element name="basket-of">
          </xs:sequence>
        </xs:complexType>
      <xs:key name="basketKey">
        <xs:selector xpath="/bookstore/basket">
        <xs:field xpath="basketid">
      </xs:key>
    </xs:schema>
```

10.16 Write queries in XQuery on the bibliography DTD fragment in Figure 10.15, 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.
- **d.** Find all books that contain the word "database" in their title and the word "Hank" in an author's name (whether first or last).

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
```

d. Find all books that contain the word "database" in their title and the word "Hank" in an author's name (whether first or last).

```
for $a in /bibliography/book/
where $b in $a satisfies
      (contains($b/title,"database") AND
      (contains($b/author(@last_name),"Hank")
            OR contains($b/author(@fiest_name),"Hank"))
return $a
```

10.17 Give a relational mapping of the XML purchase order schema illustrated in Figure 10.2, using the approach described in Section 10.6.2.3.

Suggest how to remove redundancy in the relational schema, if item identifiers functionally determine the description and purchase and supplier names functionally determine the purchase and supplier address, respectively.

Answer: Removing redundancy:

Item ids functionally determine the desription The table 'item' can be broken down into 2 tables

```
item(item_id, quantity, price, itemlist_id)
and
item_info(item_id, description)
```

Then the item description doesnt get repeated whenever the item appears on the itemlist.

Purchaser_names functionally determine the address The table purchaser can be subdivided as shown below

```
purchaser(name, purcahseorder_id)
and
purchaser_info(name, address)
```

Supplier_names functionally determine the address The table supplier can be subdivided as shown below

supplier(name, purchaseorder_id)
and
supplier_info(name, address)

The address does'nt get repeated all the times the supplier or purchaser is involved in a purchase.

```
create table purchase_order
        (purchaseorder_id char(20),
        purchaser_name char(20),
        supplier_name char(20),
        itemlist_id char(20),
        total_cost numeric(16,2),
        payment_terms char(20),
        shipping_mode char(20),
        primary key (identifier),
        foreign key (purchaser_name) references purchaser,
        foreign key (supplier_name) references supplier,
        foreign key (itemlist_id) references itemlist)

create table purchaser
        (purchaser_name char(20),
        address char(20),
        purchaseorder_id char(20),
        primary key (name),
        foreign key (purchaseorder_id) references purchase_order)

create table supplier
        (supplier_name char(20),
        address char(20),
        purchaseorder_id char(20),
        primary key (name),
        foreign key (purchaseorder_id) references purchase_order)

create table itemlist
        (itemlist_id char(20),
        item_id char(20),
        purchaseorder_id char(20),
        primary key (itemlist_id),
        foreign key (item_id) references item,
        foreign key (purchaseorder_id) references purchase_order)

create table item
        (item_id char(20),
        description char(20),
        quantity numeric(16),
        price numeric(16,2),
        itemlist_id char(20),
        primary key (item_id),
        foreign key (itemlist_id) references itemlist)
```

10.18 Write queries in SQL/XML to convert bank data from the relational schema we have used in earlier chapters to the *bank-1* and *bank-2* XML schemas. (For the *bank-2* schema, you may assume that the customer relation has an extra attribute *customer_id*.)
Answer: Bank_1:

```
select xmlelement(name "customer",
        xmlelement(name "customer_name", c.customer_name),
        xmlelement(name "customer_street" c.customer_street),
        xmlelement(name "customer_city" c.customer_city),
        xmlagg(xmlelement(name "account",
              xmlelement(name "account_number" a.account_number),
              xmlelement(name "branch_name" a.branch_name),
              xmlelement(name "balance" a.balance)
              from account a, depositor d
              where a.account_number = d.account_number and
              d.customer_name = c.customer_name
              order by a.account_number)))
        from customer c
        group by c.customer_name
```

Bank_2:

```
select xmlelement(name "account",
        xmlattributes(account_number as account_number),
        xmlelement(name "branch_name" branch_name),
        xmlelement(name "balance" balance))
        from account

select xmlelement(name "customer",
        xmlattributes(customer_id as coustomer_id),
        xmlelement(name "customer_name" customer_name),
        xmlelement(name "customer_street" customer_street),
        xmlelement(name "customer_city" customer_city))
        from customer
```

10.19 As in Exercise 10.18, write queries to convert bank data to the *bank-1* and *bank-2* XML schemas, but this time by writing XQuery queries on the default SQL/XML database to XML mapping.
Answer: bank-1:

```
for $x in /bank_1/customer/row/
return
    <bank_1>
        <customer>
            <customer_name> {$x/customer_name} </customer_name>
            <customer_street> {$x/customer_street} </customer_street>
            <customer_city> {$x/customer_city} </customer_city>
            <account> {$x/account/row} </account>
        </customer>
    </bank_1>
```

bank-2:

```
for $x in /bank_2/account/row/
    $y in /bank_2/customer/row/
return
    <bank_2>
        <account account_number = "{$x(@account_number)}">
        {$x/*} </account>
        <customer customer_id = "{$y(@customer_id)}">
        {$y/*} </customer>
    </bank_2>
```

10.20 One way to shred an XML document is to use XQuery to convert the schema to an SQL/XML mapping of the corresponding relational schema, and then use the SQL/XML mapping in the backward direction to populate the relation.

As an illustration, give an XQuery query to convert data from the *bank-1* XML schema to the SQL/XML schema shown in Figure 10.14.

Answer:

```
for $x in /bank_1/customer/
return
    <bank_1>
        <account>
            <row> {"$x/account"} </row>
        </account>
        <customer>
            <row>
            <customer_name> {"$x/customer_name"} </customer_name>
            <customer_street> {"$x/customer_street"} </customer_street>
            <customer_city> {"$x/customer_city"} </customer_city>
            </row>
        </customer>
        <depositor>
            <row>
            <account_number> {"$x/account/account_number"} </account_number>
            <customer_name> {"$x/customer_name"} </customer_name>
            </row>
        </depositor>
    </bank_1>
```

10.21 Consider the example XML schema from Section 10.3.2, and write XQuery queries to carry out the following tasks.

a. Check if the key constraint shown in Section 10.3.2 holds.

b. Check if the keyref constraint shown in Section 10.3.2 holds.

Answer:

a. Check if the key constraint shown in Section 10.3.2 holds.

```
for $x in /bank_1/account/
    let $y = distinct-values($x/account_number)
return
    if {count($y) == {$x[count(*)]}}
        then 1
    else 0
```

b. Check if the keyref constraint shown in Section 10.3.2 holds.

```
for $x in /bank_1/depositor/
    $y in /bank_1/account/
return
    (every $p in $x satisfies
        $p/account_number = $y/account_number)
```

10.22 Consider Practice Exercise 10.7, and suppose that authors could also appear as top-level elements. What change would have to be done to the relational schema?

Answer:

Author id can be added as an attrribute to the author top element. It can be used to refer to the author in the book and article topelements and also keep track of the order.

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.

Exercises

11.8 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.

11.9 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!

11.10 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.

11.11 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.

11.12 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.

11.13 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.

11.14 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.

11.15 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.

11.16 If you have data that should not be lost on disk failure, and the data are write intensive, how would you store the data?
Answer: There are several possibilities, each with different performance and cost implications. A **RAID** array can handle the failure of a single drive (two drives in the case of RAID 6) without data loss, and is relatively inexpensive. Another possibility would be to use solid-state storage such as battery backed **NV-RAM**, albeit at a much higher cost per megabyte.

11.17 In earlier generation disks the number of sectors per track was the same across all tracks. Current generation disks have more sectors per track on outer tracks, and fewer sectors per track on inner tracks (since they are shorter in length). What is the effect of such a change on each of the three main indicators of disk speed?
Answer: The main performance effect of storing more sectors on the outer tracks and fewer sectors on the inner tracks is that the disk's **data-transfer** rate will be greater on the outer tracks than the inner tracks. This is because the disk spins at a constant rate, so more sectors pass underneath the drive head in a given amount of time when the arm is positioned on an outer track than when on an inner track.

11.18 Standard buffer managers assume each page is of the same size and costs the same to read. Consider a buffer manager that, instead of LRU, uses the rate of reference to objects, that is, how often an object has been accessed in the last n seconds. Suppose we want to store in the buffer objects of varying sizes, and varying read costs (such as Web pages, whose read cost depends on the site from which they are fetched). Suggest how a buffer manager may choose which page to evict from the buffer.
Answer: A good solution would make use of a *priority queue* to evict pages, where the priority (p) is ordered by the *expected cost* of re-reading a page given it's past access frequency (f) and it's re-read cost (c).

$$p = f * c$$

The buffer manager should choose to evict the page with the lowest expected cost of re-reading.

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) and bitmap indexing (Section 12.9) may be omitted if desired.

Exercises

12.13 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.

12.14 What is the difference between a clustering index and a secondary index?

Answer: The clustering index is on the field which specifies the sequential order of the file. There can be only one clustering index while there can be many secondary indices.

12.15 For each B^+-tree of Practice Exercise 12.3, 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 the structure provided by the solution to Practice Exercise 12.3a:

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 the structure provided by the solution to Practice Exercise 12.3b:

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 the structure provided by the solution to Practice Exercise 12.3c:

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.

12.16 he solution presented in Section 12.5.3 to deal with non-unique search keys added an extra attribute to the search key. What effect does this change have on the height of the B^+-tree?

Answer: The resultant B-tree's extended search key is unique. This results in more number of nodes. A single node (which points to mutiple records with the same key) in the original tree may correspond to multiple nodes in the

result tree. Depending on how they are organized the height of the tree may increase; it might be more than that of the original tree.

12.17 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.

12.18 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.

12.19 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.

12.20 Suppose there is a relation $R(A, B, C)$, with a B^+-tree index with search key (A, B).

- **a.** What is the worst case cost of finding records satisfying $10 < A < 50$ using this index, in terms of the number of records retrieved n_1 and the height h of the tree?

b. What is the worst case cost of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$ using this index, in terms of the number of records n_2 that satisfy this selection, as well as n_1 and h defined above.

c. Under what conditions on n_1 and n_2 would the index be an efficient way of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$.

Answer:

a. What is the worst case cost of finding records satisfying $10 < A < 50$ using this index, in terms of the number of records retrieved n_1 and the height h of the tree?

This query does not correspond to a range query on the search key as the condition on the first attribute if the search key is a comparison condition. It looks up records which have the value of A between 10 and 50. However, each record is likely to be in a different block, because of the ordering of records in the file, leading to many I/O operation. In the worst case, for each record, it needs to traverse the whole tree (cost is h), so the total cost is $n_1 * h$.

b. What is the worst case cost of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$ using this index, in terms of the number of records n_2 that satisfy this selection, as well as n_1 and h defined above.

This query can be answered by using an ordered index on the search key (A, B). For each value of A this is between 10 and 50, the system located records with B value between 5 and 10. However, each record could is likely to be in a different disk block. This amounts to executing the query based on the condition on A, this costs $n_1 * h$. Then these records are checked to see if the condition on B is satisfied. So, the total cost in the worst case is $n_1 * h$.

c. Under what conditions on n_1 and n_2 would the index be an efficient way of finding records satisfying $10 < A < 50 \wedge 5 < B < 10$.

n_1 records satisfy the first condition and n_2 records satisfy the second condition. When both the conditions are queried, n_1 records are output in the first stage. So, in the case where $n_1 = n_2$, no extra records are output in the furst stage. Otherwise, the records which dont satisfy the second condition are also output with an additional cost of h each (worst case).

12.21 Suppose you have to create a B^+-tree index on a large number of names, where the maximum size of a name may be quite large (say 40 charac ters) and the average name is itself large (say 10 characters). Explain how prefix compression can be used to maximize the average fanout of internal nodes.

Answer: There arises 2 problems in the given scenario. The first problem is names can be of vairable length. The second problem is names can be long (maximum is 40 characters), leading to a low fanout and a correspondingly increased tree height. With variable-length search keys, different nodes can have different fanouts even if they are full. The fanout of nodes can be increased bu using a technique called prefix compression. With prefix compression, the entire search key value is not stored at internal nodes. Only a prefix of each

search key which is sufficeint to distinguish between the key values in the subtrees that it seperates. The full name can be stored in the leaf nodes, this way we dont lose any information and also maximize the average fanout of internal nodes.

12.22 Why might the leaf nodes of a B^+-tree file organization lose sequentiality? Suggest how the file organization may be reorganized to restore sequentiality.
Answer: In a B^+-tree index or file organization, leaf nodes that are adjacent to each other in the tree may be located at different places on disk. When a file organization is newly created on a set of records, it is possible to allocate blocks that are mostly contiguous on disk to leafs nodes that are contiguous in the tree. As insertions and deletions occur on the tree, sequentiality is increasingly lost, and sequential access has to wait for disk seeks increasingly often. A way to solve this problem might be to rebuild the index to retsore sequentiality.

12.23 Suppose a relation is stored in a B^+-tree file organization. Suppose secondary indices stored record identifiers that are pointers to records on disk.

a. What would be the effect on the secondary indices if a page split happens in the file organization?
b. What would be the cost of updating all affected records in a secondary index?
c. How does using the search key of the file organization as a logical record identifier solve this problem?
d. What is the extra cost due to the use of such logical record identifiers?

Answer:

12.24 When a leaf page is split in a B^+-tree file organization, a number of records are moved to a new page. In such cases, all secondary indices that store pointers to the relocated records would have to be updated, even though the values in the records may not have changed.

12.25 Each leaf page may contain a fairly large number of records, and each of them may be in different locations on each secondary index. Thus, a leaf-page split may require tens or even hundreds of I/O operations to update all affected secondary indices, making it a very expensive operation.

12.26 One solution might be in secondary indices, in place of pointers to the indexed records, we store the values of the primary-index serach key attributes. Relocation of records because of leaf-page splits then does not require any update on any secondary index.

12.27 Locating a record using the secondary index now requires two steps: First we use the secondary index to find the primary index search-key values, and then we use the primary index to find the corresponding records. This approach reduces the cost of index update due to file reorganization, although it increases the cost of accesing data using a secondary index.

12.28 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*.

12.29 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.

12.30 Our description of static hashing assumes that a large contiguous stretch of disk blocks can be allocated to a static hash table. Suppose you can allocate only C contiguous blocks. Suggest how to implement the hash table, if it can be much larger than C blocks. Access to a block should still be efficient.
Answer: A separate list/table as shown below can be created.
Starting address of first set of C blocks
C
Starting address of next set of C blocks
$2C$
and so on

Desired block address = Starting adress (from the table depending on the block number) + blocksize * (blocknumber $\%$ C)

For each set of C blocks, a single entry is added to the table. In this case, locating a block requires 2 steps: First we use the block number to find the actual block address, and then we can access the desired block.

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 2, 11, and 12.

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

13.10 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.

13.11 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.

13.12 Estimate the number of block accesses required by your solution to Practice Exercise 13.11 for $r_1 \bowtie r_2$, where r_1 and r_2 are as defined in Exercise 13.3.
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 $\lceil n_o/n_2 \rceil$.

13.13 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.
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 $t_r \ \mathbin{⟕}\ t_s$. To get the natural right outer join $t_r \ \mathbin{⟖}\ t_s$, we can keep a boolean flag with each

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

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.

13.14 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 **IndexedNLJoin::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

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

$done_r$ 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
```

13.15 Pipelining is used to avoid writing intermediate results to disk. Suppose you need to sort relation r using sort–merge and merge-join the result with an already sorted relation s.

a. Describe how the output of the sort of r can be pipelined to the merge join without being written back to disk.

b. The same idea is applicable even if both inputs to the merge-join are the outputs of sort–merge operations. However, the available memory has to be shared between the two merge operations (the merge-join algorithm itself needs very little memory). What is the effect of having to share memory on the cost of each sort–merge operation.

Answer:

a. Using pipelining, output from the sorting operation on r is written to a buffer B. When B is full, the merge-join processes tuples from B, joining them with tuples from s until B is empty. At this point, the sorting operation is resumed and B is refilled. This process continues until the merge-join is complete.

b. If the sort–merge operations are run in parallel and memory is shared equally between the two, each operation will have only $M/2$ frames for its memory buffer. This may increase the number of runs required to merge the data.

13.16 Suppose you have to compute ${}_{A}\mathcal{G}_{sum(C)}(r)$ as well as ${}_{A,B}\mathcal{G}_{sum(C)}(r)$. Describe how to compute these together using a single sorting of r.
Answer: Run the sorting operation on r, grouping by (A, B), as required for the second result. When evaluating the sum aggregate, keep running totals for both the (A, B) grouping as well as for just the A grouping.

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.

Exercises

14.11 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.

14.12 Show how to derive the following equivalences by a sequence of transformations using the equivalence rules in Section 14.2.1.

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 $\sigma_{\theta_1}(E_1 \bowtie_{\theta_3} (\sigma_{\theta_2}(E_2)))$.

14.13 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 14.2.1 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 14.2.1 is not complete. The expressions $\sigma_{3=5}(r)$ and $\{\ \}$ are equivalent, but this cannot be shown by using just these rules.

14.14 Explain how to use a histogram to estimate the size of a selection of the form $\sigma_{A \leq v}(r)$.
Answer: Suppose the histogram H storing the distribution of values in r is divided into ranges $r_1, \ldots, r_n$. For each range r_i with low value $r_{i:low}$ and high value $r_{i:high}$, if $r_{i:high}$ is less than v, we add the number of tuples given by

$$H(r_i)$$

to the estimated total. If $v < r_{i:high}$ and $v >= r_{i:low}$, we assume that values within r_i are uniformly distributed and we add

$$H(r_i) * \frac{v - r_{i:low}}{r_{i:high} - r_{i:low}}$$

to the estimated total.

14.15 Suppose two relations r and s have histograms on attributes $r.A$ and $s.A$, respectively, but with different ranges. Suggest how to use the histograms to estimate the size of $r \bowtie s$. Hint: Split the ranges of each histogram further.
Answer: Find the largest unit u that evenly divides the range size of both histograms. Divide each histogram into ranges of size u, assuming that values within a range are uniformly distributed. Then compute the estimated join size using the technique for histograms with equal range sizes.

14.16 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 $v = r \mathbin{⟕} s$, when a set of tuples i_r is inserted in r, we add the tuples $i_r \mathbin{⟕} s$ to the view. When i_r is deleted from r, we delete $i_r \mathbin{⟕} s$ 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 $r \mathbin{⟕} s$) 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.

14.17 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.

14.18 Suppose you want to get answers to $r \bowtie s$ sorted on an attribute of r, and want only the top K answers for some relatively small K. Give a good way of evaluating the query

a. When the join is on a foreign key of r referencing s.
b. When the join is not on a foreign key.

Answer:

a. Sort r and collect the top K tuples. These tuples are guaranteed to be contained in $r \bowtie s$ since the join is on a foreign key of r referencing s.
b. Execute $r \bowtie s$ using a standard join algorithm until the first K results have been found. After K tuples have been computed in the result set, continue executing the join but immediately discard any tuples from r that have attribue values less than all of the tuples in the result set. If a newly joined tuple t has an attribute value greater than at least one of the tuples in the result set, replace the lowest-valued tuple in the result set with t.

14.19 Consider a relation $r(A, B, C)$, with an index on attribute A. Give an example of a query that can be answered by using the index only, without looking at the tuples in the relation. (Query plans that use only the index, without accessing the actual relation, are called *index-only* plans.)
Answer: Any query that only involves the attribute A of r can be executed by only using the index. For example, the query

```
select sum(A)
from r
```

only needs to use the values of A, and thus does not need to look at r.

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 25), we use the stronger condition of serializability as a requirement on concurrent transactions.

Exercises

15.8 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.

15.9 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.

15.10 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.

15.11 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.

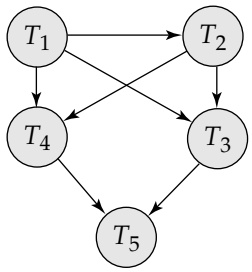


Figure 15.18. Precedence graph.

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).

15.12 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.

15.13 Why do database systems support concurrent execution of transactions, in spite of the extra programming effort needed to ensure that concurrent execution does not cause any problems?
Answer: Transaction-processing systems usually allow multiple transactions to run concurrently. It is far easier to insiat that transactions run serially. However there are two good reasons for allowing concurrency:

- Improved throughput and resource utilization. A transaction my involve I/O activity, CPU activity. The CPU and the disk in a computer system can operate in parallel. This can be exploited to run multiple transactions in parallel. For example, while a read or write on behalf of one transaction is in progress on one disk, another transacation can be running in the CPU. This increases the throughput of the system.
- Reduced waiting time. If transactions run serially, a short transaction may have to wait for a preceding long transaction to complete. If the transactions are operating on different parts of the database, it is better to let them

run concurrently, sharing the CPU cycles and disk accesses among them. It reduces the unpredictable delays and the average response time.

CHAPTER 16

Concurrency Control

Exercises

16.19 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.

16.20 Most implementations of database systems use strict two-phase locking. Suggest three reasons for the popularity of this protocol.

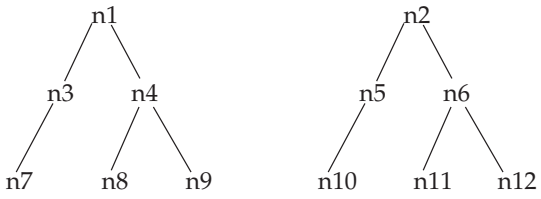
Answer: It is relatively simple to implement, imposes low rollback overhead because of cascadeless schedules, and usually allows an acceptable level of concurrency.

16.21 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.

16.22 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.

16.23 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.

16.24 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.

16.25 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 marks*
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.

16.26 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.

16.27 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.

16.28 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.

16.29 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.

16.30 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.

16.31 Give example schedules to show that with key-value locking, if any of lookup, insert, or delete do not lock the next-key value, the phantom phenomemon could go undetected.
Answer: In the next-key locking technique, every index lookup or insert or delete must not only the keys found within the range (or the single key, in case of a point lookup) but also the next-key value- that is, the key value greater

than the last key value that was within the range. Thus, if a transaction attempts to insert a value that was within the range of the index lookup of another transaction, the two transactions would conflict ion the key value next to the inserted key value. The next-key value should be locked to ensure that conflicts with subsequent range lookups of other queries are detected, thereby detecting phantom phenomenon.

16.32 If many transactions update a common item (e.g., the cash balance at a br anch), and private items (e.g., individual account balances). Explain how you can increase concurrency (and throughput) by ordering the operations of the transaction.

Answer: The private items can be updated by the individual transacations independently. They can acquire the exclusive locks for the private items (as no other transaction needs it) and update the data items. But the exclusive lock for the common item is shared among all the transactions. The common item should be locked before the transaction decides to update it. And when it holds the lock for the common item, all other transactions should wait till its released. But inorder that the common item is updated correctly, the transaction should follow a certain pattern. A transacation can update its private item as and when it requires, but before updating the private item again, the common item should be updated. So, essentially the private and the common items should be accessed alternately, otherwise the private item's update will not be reflected in the common item.

a. No possibility of deadlock and no starvation. The lock for the common item should be granted based on the time of requests.

b. The schedule is serializable.

16.33 Consider the following locking protocol: All items are numbered, and once an item is unlocked, only higher numbered items may be locked. Locks may be released at any time. Only X-locks are used.

Show by an example that this protocol does not guarantee serializability.

Answer: We have 2 transactions, T_1 and T_2. Consider the following legal schedule:

T_1	T_2
lock(A)	
write(A)	
unlock(A)	
	lock(A)
	read(A)
	lock(B)
	write(B)
	unlock(B)
lock(B)	
read(B)	
unlock(B)	

Explanation: In the given example schedule, lets assume A is a hiher numbered item then B.

a. T_i executes write(A) before T_j executes read(A). So, there's a edge T_i->T_j.
b. T_j executes write(B) before T_i executes read(A). So, there's a edge T_j->T_i.

There's a cycle in the graph which means the given schedule is not conflict serializable.

CHAPTER 17

Recovery System

Exercises

17.7 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.

17.8 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.

17.9 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 transaction in stable storage.

17.10 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.

17.11 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.

17.12 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.

17.13 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.

17.14 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.

17.15 The Oracle database system uses undo log records to provide a snapshot view of the database to read-only transactions. The snapshot view reflects updates of all transactions that had committed when the read-only transaction started; updates of all other transactions are not visible to the read-only transactions.

Describe a scheme for buffer handling whereby read-only transactions are given a snapshot view of pages in the buffer. Include details of how to use the log to generate the snapshot view, assuming that the advanced recovery algorithm is used. Assume for simplicity that a logical operation and its undo affect only a single page.
Answer: First, determine if a transaction is currently modifying the buffer. If not, then return the current contents of the buffer. Otherwise, examine the

records in the undo log pertaining to this buffer. Make a copy of the buffer, then for each relevant operation in the undo log, apply the operation to the buffer copy starting with the most recent operation and working backwards until the point at which the modifying transaction began. Finally, return the buffer copy as the snapshot buffer.

CHAPTER 18

Data Analysis and Mining

Exercises

18.8 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 calculate 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*

18.9 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 18.4. 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.

18.10 Given relation $r(a, b, c)$, Show how to use the extended SQL features to generate a histogram of c 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**(*c*)
from (**select** *c*, **ntile**(20) **over** (**order by** (*a*)) **as** *tile20*
from *r*) **as** *s*
groupby *tile20*

18.11 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)
)
```

18.12 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: Figure 18.1 shows one possible decision tree for the data. Using the Gini purity metric, the purity of the initial data set is

$$1 - \sum_{i-1}^{k} p_i^2 = 1 - ((\frac{2}{9})^2 + (\frac{5}{9})^2 + (\frac{2}{9})^2) = 0.595259$$

The first branch splits on $B <= 2$, giving a purity score of $1 - 1^2 = 0$ for those attributes with $B <= 2$ (all are classified as a), and a purity score of

$$1 - ((\frac{2}{7})^2 + (\frac{5}{7})^2) = 0.40816$$

for the remaining items. The weighted purity of the entire set is

$$\frac{2}{9} * 0 + \frac{7}{9} * 0.40816 = 0.31746$$

The information gain from this split is $0.595259 - 0.31746 = 0.27513$.

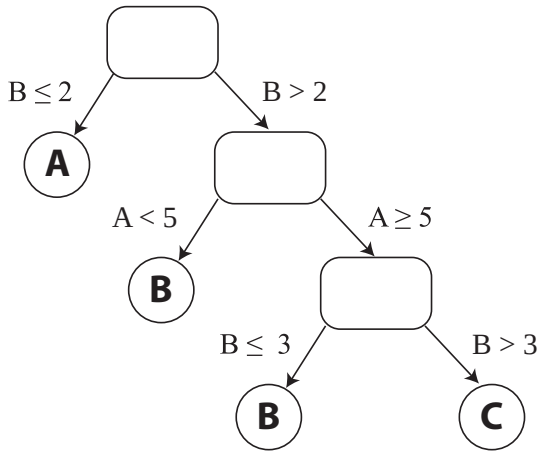


Figure 18.1 Decision tree for data on relation $r(A, B, C)$

Next, we split on $A < 5$. The 4 data items with $A < 5$ all have class b, and thus have purity 0. The remaining 3 items have purity

$$1 - ((\frac{1}{3})^2 + (\frac{2}{3})^2) = 0.44444$$

The weighted purity of these sets is

$$\frac{4}{7} * 0 + \frac{3}{7} * 0.44444 = 0.19048$$

The information gain from the second split is $0.40816 - 0.19048 = 0.21769$.

Finally, we split on $B <= 3$. One data item satisfies this predicate and has class b. The other two items both have class c. The purity of these two sets is 0. The information gain from the final split is $0.21769 - 0 = 0.21769$.

18.13 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%

18.14 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.

18.15 Create a small example of a set of transactions showing that that although many transactions contain two items, that is the itemset containing the two items has a high support, purchase of one of the items may have a negative

correlation with purchase of the other.
Answer: The following set of transactions involve fruit purchases:

Transaction_ID	Item
T-1	orange
T-1	banana
T-1	apple
T-2	orange
T-2	banana
T-3	orange
T-3	apple
T-4	orange
T-4	banana
T-4	grapes
T-5	banana
T-5	apple
T-6	banana
T-6	grapes

Consider the association rule

$$orange \Rightarrow banana$$

This rule is satisfied in 3 out of the 6 transactions, so the support value is 50 percent. However, the correlation between purchasing oranges and purchasing bananas in this data set is -0.32.

18.16 The organization of parts, chapters, sections and subsections in a book is related to clustering. Explain why, and to what form of clustering.
Answer: The organization of a book's content is a form of **hierarchical clustering**. Contents within a single subsection are closely related, whereas different parts of a book cover a more diverse range of topics.

18.17 Suggest how predictive mining techniques can be used by a sports team, using your favorite sport as an example.
Answer: Given the large amount of statistics collected during and about sporting events, there are many ways a sports team can make use of predictive data mining:

- Some players may be more effective in certain situations or environments, so data mining can predict when each player should be used.
- Specific strategies may be more effective against certain teams or during certain situations in the game.
- Predictive mining can estimate the outcome of a match beforehand, information which could be useful to a team before entering a tournament.

CHAPTER 19

Information Retrieval

Exercises

19.6 Using a simple definition of term frequency as the number of occurrences of the term in a document, give the TF-IDF scores of each term in the set of documents consisting of this and the next exercise.
Answer:

Term frequency TF(d, t) = log(1 + n(d, t)/n(d))
where n(d, t) denotes the number of occurences of term t in the document d and n(d) denotes the number of terms in the document.

using - log$(1 + 1/75)$
a - log$(1 + 5/75)$
simple - log$(1 + 1/75)$
definition - log$(1 + 1/75)$
of - log$(1 + 6/75)$
term - log$(1 + 3/75)$
frequency - log$(1 + 1/75)$
as - log$(1 + 1/75)$
the - log$(1 + 1/75)$
number - log$(1 + 7/75)$
occurences - log$(1 + 1/75)$
in - log$(1 + 2/75)$
document - log$(1 + 1/75)$
give - log$(1 + 1/75)$
TFIDF - log$(1 + 1/75)$
scores - log$(1 + 1/75)$
each - log$(1 + 3/75)$

set - log$(1 + 1/75)$
documents - log$(1 + 3/75)$
consisting - log$(1 + 1/75)$
this - log$(1 + 1/75)$
and - log$(1 + 2/75)$
next - log$(1 + 1/75)$
exercise - log$(1 + 1/75)$
create - log$(1 + 1/75)$
small - log$(1 + 2/75)$
example - log$(1 + 1/75)$
4 - log$(1 + 1/75)$
with - log$(1 + 1/75)$
PageRank - log$(1 + 3/75)$
inverted - log$(1 + 1/75)$
lists - log$(1 + 1/75)$
sorted - log$(1 + 1/75)$
by - log$(1 + 1/75)$
you - log$(1 + 1/75)$
do - log$(1 + 1/75)$
not - log$(1 + 1/75)$
need - log$(1 + 1/75)$
to - log$(1 + 1/75)$
compute - log$(1 + 1/75)$
just - log$(1 + 1/75)$
assume - log$(1 + 1/75)$
some - log$(1 + 1/75)$
values - log$(1 + 1/75)$
page - log$(1 + 1/75)$

19.7 Create a small example of a 4 small documents each with a PageRank, and create inverted lists for the documents sorted by the PageRank. You do not need to compute PageRank, just assume some values for each page.
Answer: Given 4 documents - A, B, C, D where the PageRanks are decreasing in that order, which means A has the highest PageRank and D has the lowest PageRank. We have, pages that are pointed to from more web pages have higher PageRank. Similarly, pages pointed to by Web pages with a high PageRank will also have a higher PageRank.
One way of creating an inverted list is:

a. B, C, D all point to A. $A <- B$, $A <- C$, $A <- D$.
b. A points to B. $B <- A$.
c. B points to C. $C <- B$.
d. C points to D, $D <- C$.

19.8 Suppose you wish to perform keyword querying on a set of tuples in a database, where each tuple has only a few attributes, each containing only a few words.

Does the concept of term frequency make sense in this context? And that of inverse document frequency? Explain your answer. Also suggest how you can define the similarity of two tuples using TF-IDF concepts.
Answer: Term frequency is the logarithm of the number of occurences of the term divided by the number of terms in the document. When it comes to small databases with few attributes, each containing only a few words, the concept of term frequency may not make sense. The relevance of a term may not depend on the number of occurences of the term, and also when the domain is very small the logarithmic increase we used in the term frequency many not be a good indicator.
The inverse document frequency which is the inverse of the number of documents that contain this term may not also be very relavant in this case. For example, the primary key value and some other key may be having the inverse document frequency of 1, but we cant assume their weights to be equal.
The similarity of the two tuples can be measured by the $cosinesimilarity$ metric. But one major difference is only values that belong to the same attribute should be considered. Two different attributes from two tuples may be having the same value, but that doesnt increase the similarity factor.

19.9 Web sites that want to get some publicity can join a Web ring, where they create links to other sites in the ring, in exchange for other sites in the ring creating links to their site. What is the effect of such rings on popularity ranking techniques such as PageRank?
Answer: PageRank is a measure of popularity of a page based on the popularity of the pages that link to the page. It may be noted that the pages that are pointed to from many Web pages are more likely tobe visited, and thus will have a higer PageRank. Similarly, pages pointed by Web pages with a high Pagerank will also have a higher probability of being visited, and thus will have a higher PageRank. In the given scenario where Web sites join a Web ring and create links to other sites, the PageRank of all the pages increases. The number of links referencing to each page increases, which only increases the PageRank.

19.10 The Google search engine provide a feature whereby Web sites can display advertisements supplied by Google. The advertisements supplied are based on the contents of the page. Suggest how Google might choose which advertisements to supply for a page, given the page contents.
Answer: Google might use the concepts in similarity based retrieval. Here, they can give the system a document A and the set of advertisements B, and ask the system to retrieve adverstisements that are similar to A. One appraoch is to find k terms in A with highest values of TF(A,t) * IDF(t), and to use to these k terms as a query to find relevance of other documents. The metric '$cosinesimilarity$' can also be used to determine which advertisements to supply for a page, given the page contents.

19.11 One way to create a keyword-specific version of PageRank is to modify the random jump such that a jump is only possible to pages containing the keyword.

Thus pages that do not contain the keyword but are close (in terms of links) to pages that contain the keyword also get a non-zero rank for that keyword.

a. Give equations defining such a keyword-specific version of PageRank.
b. Give a formula for computing the relevance of a page to a query containing multiple keywords.

Answer:

a. Give equations defining such a keyword-specific version of PageRank.

$$P[j] = \delta/N_i + (1-\delta) * \sum_{i=1}^{N} (T[i,j] * P]i])$$

where δ is a constant between 0 and 1, N is the number of pages, N_i is the number of pages containing the keyword; δ represents the probability of a step in the walk being a jump to a page containing the keyword.

b. Give a formula for computing the relevance of a page to a query containing multiple keywords.

The relevance of a document of a document to a query comtaining multiple keywords is estimated by combining by adding the relevance measures of the page to each keyword. Some weights also might be considered.

$$r(d,Q) = \sum_{t=1}^{n} TF(d,t) * IDF(t)$$

where Q is the set of keywords of size n, TF is the $termfrequency$ and IDf is the $inversedocumentfrequency$. This measure can be further refined if the user is permitted to specify weights $w(t)$ for terms in the query, in which case the user specified weights are also taken into account by multiplying $TF(d,t)$ by $w(t)$ in the above formula.

19.12 The idea of popularity ranking using hyperlinks can be extended to relational and XML data, using foreign key and IDREF edges in place of hyperlinks. Suggest how such a ranking scheme may be of value in the following applications.

a. A bibliographic database, which has links from articles to authors of the articles and links from each article to every article that it references.
b. A sales database which has links from each sales record to the items that were sold.

Also suggest why prestige ranking can give less than meaningful results in a movie database that records which actor has acted in which movies.

Answer:

a. A bibliographic database, which has links from articles to authors of the articles and links from each article to every article that it references.

This helps us in ranking articles according to their popularity. If an article is referenced by many articles, then its more popular, so each article

has a rank associated with it. and we could also find the authors for those articles.

b. A sales database which has links from each sales record to the items that were sold.

This helps us in determining wich items are the most popular. If the item is referenced by many sales records, then its more popular. So, ranking the database helps in determining the popularity of the items that were sold.

A movie which has many actors associated with it is deemed to be more popular when prestige ranking is taken into account. Same goes with the actor as well. But that may not be true in real life, the popularity of a movie or that of an actor cannot be determined by ranking the movie lists which map the actors to the movies.

19.13 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: False drop - A few relevant documents may not be retrieved.
False positive - A few irrelevant documents may be retrieved.
It is acceptable to have false positives but not any false drops when it is essential that no relevant information is to missed because by permitting false positive; the system can later filter the results away later by looking at the keywords than they actually contain, but by permitting false drops, some relvant information is missed out. By allowing false positives and not allowing false drops, no relevant information is missed out.

CHAPTER 20

Database-System Architectures

Exercises

20.6 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.

20.7 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.

20.8 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.

20.9 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.

20.10 Database systems are typically implemented as a set of processes (or threads) sharing a shared memory area.

a. How is access to the shared memory area controlled?

b. Is two-phase locking appropriate for serializing access to the data structures in shared memory? Explain your answer.

Answer:

a. A locking system is necessary to control access to the shared data structures. Since many database transactions only involve reading from a data structure, **reader-writer** locks should be used, allowing multiple processes to concurrently read from a data structure by using the lock in **shared** mode. A process that wishes to modify the data structure needs to obtain an **exclusive** lock on the data structure which prohibits concurrent access from other reading or writing processes.

b. Two-phase locking is an important component of serializing access to shared memory in a multi-process system, since it guarantees that transactions can in fact be serialized. However, two-phase locking by itself does not guarantee freedom from deadlock. Thus, a deadlock avoidance algorithm

such as the Banker's algorithm or a deadlock detection and recovery algorithm should also be used.

20.11 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.

20.12 Processor speeds have been increasing much faster than memory access speeds. What impact does this have on the number of processors that can effectively share a common memory?

Answer: As processor speed growth continues to outpace memory performance, fewer and fewer processors will effectively be able to share memory over a common bus. Each additional processor will have to spend a greater portion of its time waiting for access to memory due to contention from the other processors. Having relatively slower memory will amplify this effect.

20.13 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.

CHAPTER 21

Parallel Databases

Exercises

21.7 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.

21.8 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.

21.9 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.

21.10 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.

21.11 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 despite the use of parallelism.

21.12 Suppose you wished to handle a workload consisting of a large number of small transactions by using shared nothing parallelism.

a. Is intraquery parallelism required in such a situation? If not, why, and what form of parallelism is appropriate?

b. What form of skew would be of significance with such a workload?

c. Suppose most transactions accessed one *account* record, which includes an account type attribute, and an associated *account_type_master* record, which

provides information about the account type. How would you partition and/or replicate data to speed up transactions? You may assume that the *account_type_master* relation is rarely updated.

Answer:

a. Intraquery parallelism is probably not appropriate for this situation. Since each individual transaction is small, the overhead of parallelizing each query may exceed the potential benefits. Interquery parallelism would be a better choice, allowing many transactions to run in parallel.

b. Partition skew can be a performance issue in this type of system, especially with the use of shared-nothing parallelism. A load imbalance amongst the processors of the distributed system can significant reduce the speedup gained by parallel execution. For example, if all transactions happen to involve only the data in a single partition, the processors not associated with that partition will not be used at all.

c. Since *account_type_master* is rarely updated, it can be replicated in entirety across all nodes. If the *account* relation is updated frequently and accesses are well-distributed, it should be partitioned across nodes.

CHAPTER 22

Distributed Databases

Exercises

22.13 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.

22.14 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.

22.15 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.

22.16 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.

22.17 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.

22.18 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: PostgreSQL does not have built-in support for replication. However, there are several external projects that add replication support to the database engine.

- **Slony-I** adds basic master-slave replication functionality to PostgreSQL with updates to the replicated databases performed lazily. A consistent view of the replicated data is provided by preserving transactions across the replicated sites. Thus, the replicated sites will always have a consistent, although potentially older, version of the data.
- **Postgres-R** extends PostgreSQL to add synchronous replication, thus avoiding the consistency issues of performing lazy updates.

22.19 Discuss the advantages and disadvantages of the two methods that we presented in Section 22.5.2 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.

22.20 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 $\Pi_{plant\_number}\ (\sigma_{machine\_number=1130}\ (machine))$ 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

$$\Pi_{plant\_number}\ (\sigma_{type=\text{“milling machine”}}\ (machine))$$ at Armonk.

c. i. Perform $\sigma_{plant\_number\ =\ x}\ (machine)$ 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 *employee* fragment at that site. As a result the second strategy will result in less data shipping, and will be more efficient.

22.21 For each of the strategies of Exercise 22.20, 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 strategy 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 22.20 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.

22.22 Is the expression $r_i \ltimes r_j$ 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 Exercise 22.11, in which $r \ltimes s \neq s \ltimes r$. $r \ltimes s$ was given in 22.11, 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.

22.23 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 23

Advanced Application Development

Exercises

23.6 Find out what all performance information your favorite database system provides. Look for at least the following: what queries are currently executing or executed recently, what resources each of them consumed (CPU and I/O), what fraction of page requests resulted in buffer misses (for each query, if available), and what locks have a high degree of contention. You may also be able to get information about CPU and I/O utilization from the operating system.

Answer:

Postgresql: The $EXPLAIN$ command lets us see what query plan the system creates for any query. The numbers that are currently quoted by $EXPLAIN$ are:

a. Estimated start-up cost
b. Estimated total cost
c. Estimated number of rows output by this plan node
d. Estimated average width of rows

SQL: There is a Microsoft tool called $SQLProfiler$. The data is logged, and then the performance can be monitored. The following performance counters can be logged.

a. Memory
b. Physical Disk
c. Process
d. Processor
e. SQLServer:Access Methods
f. SQLServer:Buffer Manager
g. SQLServer:Cache Manager
h. SQLServer:Databases

i. SQLServer:General Statistics
j. SQLServer:Latches
k. SQLServer:Locks
l. SQLServer:Memory Manager
m. SQLServer:SQL Statistics
n. SQLServer:SQL Settable

23.7 **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.

23.8 When carrying out performance tuning, should you try to tune your hardware (by adding disks or memory) first, or should you try to tune your transactions (by adding indices or materialized views) first. Explain your answer.
Answer: The 3 levels of tuning - hardware, database system parameters, schema and transactions - interact with one another; so we must consider them together when tuning a system. For example, tuning at the transaction level may result in the hardware bottleneck changing from the disk system to the CPU, or vice versa.

23.9 Suppose that your application has transactions that each access and update a single tuple in a very large relation stored in a B^+-tree file organization. Assume that all internal nodes of the B^+-tree are in memory, but only a very small fraction of the leaf pages can fit in memory. Explain how to calculate the minimum number of disks required to support a workload of 1000 transactions per second. Also calculate the required number of disks, using values for disk parameters given in Section 11.2.
Answer: Given that all internal nodes of the B^+-tree are in memory, and only a very small fraction of the leaf pages can fit in memory. We can deduce that each I/O transaction that access and update a single tuple requires just 1 I/O operation. The disk with the parameters given in the chapter would support a little under 100 random-access I/O operations of 4 kilbytes each per second. So, number of disks needed to support a workload of 1000 transactions is 1000/100 = 10 disks.
The disk parameters given in Section 11.2 are almost the same as the values in the current chapter. So, the number of disks required will be around 10 in this case also.

23.10 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.

23.11 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.

23.12 List at least 4 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.

23.13 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.

23.14 Explain what application characteristics would help you decide which of TPCC, TPCH, or TPCR best models the application.
Answer: Depending on the application characterictics, different benchmarks are used to model it.
The TPCC benchmark is widely used for transaction processing. It is appropriate for applications which concentrate on the main activities in an order-entry environment, such as entering and delivering orders, recording payments, checking status of orders, and monitoring levels of stock.
The TPCH (H represents $adhoc$) benchmark models the applications which prohibit materialized views and other redundant information, and permits indices only on primary and foriegn keys. This benchmark models ad-hoc querying where the queries are not known beforehand.

The TPCR (R represents for *reporting*) models the applications which has queries, inserts, updates, and deletes. The application is permitted to use materialized views and other redundant information.

CHAPTER 24

Advanced Data Types and New Applications

Exercises

24.9 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* $\xrightarrow{\tau}$ *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.

24.10 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).

24.11 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: PostgreSQL includes support for R-tree indices over spatial data, as well as a number of built-in geometric data types (points, boxes, circles, lines, and paths) to represent spatial data, and functions to manipulate this data.

a.

```
create table restaurants (
    name varchar(30),
    location point,
    cusine varchar(30),
    price int)
```

b. Assume your house is at coordinates $(21.5, 14.2)$, and that a price value of 2 means "moderately priced".
`<->` is the PostgreSQL operator representing "distance between".

```
select name
from restaurants
where ((point '(21.5, 14.2)') <-> location) < 5.0
    and cusine = 'Indian'
    and price <= 2
```

c.

```
select r1.name, min(r1.location <-> r2.location)
from restaurants as r1, restaurants as r2
where r1.cusine = r2.cusine
    and r1.price = r2.price
group by r1.name
```

24.12 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.

24.13 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.

24.14 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.

24.15 Give an example to show that the version-vector scheme does not ensure serializability. (Hint: Use the example from Exercise 24.8, 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 25

Advanced Transaction Processing

Exercises

25.9 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.

25.10 Compare TP monitor features with those provided by Web servers supporting servlets (such servers have been nicknamed *TP-lite*).

Answer: No answer.

25.11 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: No answer.

25.12 Answer the following questions regarding electronic payment systems.

a. Explain why electronic transactions carried out using credit card numbers are insecure.
b. An alternative is to have an electronic payment gateway maintained by the credit card company, and the site receiving payment redirects customers to the gateway site to make the payment.
   i. Explain what benefits such a system offers if the gateway does not authenticate the user
   ii. Explain what further benefits are offered if the gateway has a mechanism to authenticate the user.
c. Some credit card companies offer a one-time-use credit card number as a more secure method of electronic payment. Customers connect to the credit card companys Web site to get the one-time-use number. Explain what benefit such a system offers, as compared to using regular credit card numbers. Also explain its benefits and drawbacks as compared to electronic payment gateways with authentication.
d. Does either of the above systems guarantee the same privacy that is available when payments are made in cash? Explain your answer.

Answer: No Answer

25.13 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.

25.14 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.

25.15 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.

25.16 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.

25.17 Explain the connections between a workflow and a long duration trans action.

Answer: No answer.