DATABASE MANAGEMENT SYSTEMS SOLUTIONS MANUAL

Raghu Ramakrishnan et al.
University of Wisconsin
Madison, WI, USA

Johannes Gehrke and Jeff Derstadt
Cornell University
Ithaca, NY, USA

CONTENTS

PREFACE

> It is not every question that deserves an answer.
>
> Publius Syrus, 42 B.C.

I hope that most of the questions in this book deserve an answer. The set of questions is unusually extensive, and is designed to reinforce and deepen students' understanding of the concepts covered in each chapter. There is a strong emphasis on quantitative and problem-solving type exercises.

While I wrote some of the solutions myself, most were written originally by students in the database classes at Wisconsin. I'd like to thank the many students who helped in developing and checking the solutions to the exercises; this manual would not be available without their contributions. In alphabetical order: X. Bao, S. Biao, M. Chakrabarti, C. Chan, W. Chen, N. Cheung, D. Colwell, J. Derstadt, C. Fritz, V. Ganti, J. Gehrke, G. Glass, V. Gopalakrishnan, M. Higgins, T. Jasmin, M. Krishnaprasad, Y. Lin, C. Liu, M. Lusignan, H. Modi, S. Narayanan, D. Randolph, A. Ranganathan, J. Reminga, A. Therber, M. Thomas, Q. Wang, R. Wang, Z. Wang and J. Yuan. In addition, James Harrington and Martin Reames at Wisconsin and Nina Tang at Berkeley provided especially detailed feedback.

Several students contributed to each chapter's solutions, and answers were subsequently checked by me and by other students. This manual has been in use for several semesters. I hope that it is now mostly accurate, but I'm sure **it still contains errors and omissions**. If you are a student and you do not understand a particular solution, contact your instructor; it may be that you are missing something, but it may also be that the solution is incorrect! If you discover a bug, please send me mail (`raghu@cs.wisc.edu`) and I will update the manual promptly.

The latest version of this solutions manual is distributed freely through the Web; go to the home page mentioned below to obtain a copy.

For More Information

The home page for this book is at URL:

Database Management Systems Solutions Manual

`http://www.cs.wisc.edu/~dbbook`

This page is frequently updated and contains information about the book, past and current users, and the software. This page also contains a link to all known errors in the book, the accompanying slides, and the software. *Since the solutions manual is distributed electronically, all known errors are immediately fixed and no list of errors is maintained.* Instructors are advised to visit this site periodically; they can also register at this site to be notified of important changes by email.

1

INTRODUCTION TO DATABASE SYSTEMS

Exercise 1.1 Why would you choose a database system instead of simply storing data in operating system files? When would it make sense *not* to use a database system?

Answer 1.1 A *database* is an integrated collection of data, usually so large that it has to be stored on secondary storage devices such as disks or tapes. This data can be maintained as a collection of operating system files, or stored in a *DBMS* (database management system). The advantages of using a DBMS are:

- *Data independence and efficient access.* Database application programs are independent of the details of data representation and storage. The conceptual and external schemas provide independence from physical storage decisions and logical design decisions respectively. In addition, a DBMS provides efficient storage and retrieval mechanisms, including support for very large files, index structures and query optimization.

- *Reduced application development time.* Since the DBMS provides several important functions required by applications, such as concurrency control and crash recovery, high level query facilities, etc., only application-specific code needs to be written. Even this is facilitated by suites of application development tools available from vendors for many database management systems.

- *Data integrity and security.* The view mechanism and the authorization facilities of a DBMS provide a powerful access control mechanism. Further, updates to the data that violate the semantics of the data can be detected and rejected by the DBMS if users specify the appropriate *integrity constraints.*

- *Data administration.* By providing a common umbrella for a large collection of data that is shared by several users, a DBMS facilitates maintenance and data administration tasks. A good DBA can effectively shield end-users from the chores of fine-tuning the data representation, periodic back-ups etc.

- *Concurrent access and crash recovery.* A DBMS supports the notion of a *transaction*, which is conceptually a single user's sequential program. Users can write transactions as if their programs were running in isolation against the database. The DBMS executes the actions of transactions in an interleaved fashion to obtain good performance, but schedules them in such a way as to ensure that conflicting operations are not permitted to proceed concurrently. Further, the DBMS maintains a continuous log of the changes to the data, and if there is a system crash, it can restore the database to a *transaction-consistent* state. That is, the actions of incomplete transactions are undone, so that the database state reflects only the actions of completed transactions. Thus, if each complete transaction, executing alone, maintains the consistency criteria, then the database state after recovery from a crash is consistent.

If these advantages are not important for the application at hand, using a collection of files may be a better solution because of the increased cost and overhead of purchasing and maintaining a DBMS.

Exercise 1.2 What is logical data independence and why is it important?

Answer 1.2 Answer omitted.

Exercise 1.3 Explain the difference between logical and physical data independence.

Answer 1.3 Logical data independence means that users are shielded from changes in the logical structure of the data, while physical data independence insulates users from changes in the physical storage of the data. We saw an example of logical data independence in the answer to Exercise 1.2. Consider the Students relation from that example (and now assume that it is not replaced by the two smaller relations). We could choose to store Students tuples in a heap file, with a clustered index on the sname field. Alternatively, we could choose to store it with an index on the gpa field, or to create indexes on both fields, or to store it as a file sorted by gpa. These storage alternatives are not visible to users, except in terms of improved performance, since they simply see a relation as a set of tuples. This is what is meant by physical data independence.

Exercise 1.4 Explain the difference between external, internal, and conceptual schemas. How are these different schema layers related to the concepts of logical and physical data independence?

Answer 1.4 Answer omitted.

Exercise 1.5 What are the responsibilities of a DBA? If we assume that the DBA is never interested in running his or her own queries, does the DBA still need to understand query optimization? Why?

Answer 1.5 The DBA is responsible for:

- *Designing the logical and physical schemas, as well as widely-used portions of the external schema.*
- *Security and authorization.*
- *Data availability and recovery from failures.*
- *Database tuning:* The DBA is responsible for evolving the database, in particular the conceptual and physical schemas, to ensure adequate performance as user requirements change.

A DBA needs to understand query optimization even if s/he is not interested in running his or her own queries because some of these responsibilities (database design and tuning) are related to query optimization. Unless the DBA understands the performance needs of widely used queries, and how the DBMS will optimize and execute these queries, good design and tuning decisions cannot be made.

Exercise 1.6 Scrooge McNugget wants to store information (names, addresses, descriptions of embarrassing moments, etc.) about the many ducks on his payroll. Not surprisingly, the volume of data compels him to buy a database system. To save money, he wants to buy one with the fewest possible features, and he plans to run it as a stand-alone application on his PC clone. Of course, Scrooge does not plan to share his list with anyone. Indicate which of the following DBMS features Scrooge should pay for; in each case also indicate why Scrooge should (or should not) pay for that feature in the system he buys.

1. A security facility.
2. Concurrency control.
3. Crash recovery.
4. A view mechanism.
5. A query language.

Answer 1.6 Answer omitted.

Exercise 1.7 Which of the following plays an important role in *representing* information about the real world in a database? Explain briefly.

1. The data definition language.

2. The data manipulation language.
3. The buffer manager.
4. The data model.

Answer 1.7 Let us discuss the choices in turn.

- The data definition language is important in representing information because it is used to describe external and logical schemas.
- The data manipulation language is used to access and update data; it is not important for representing the data. (Of course, the data manipulation language must be aware of how data is represented, and reflects this in the constructs that it supports.)
- The buffer manager is not very important for representation becuase it brings arbitrary disk pages into main memory, independent of any data representation.
- The data model is fundamental to representing information. The data model determines what data representation mechanisms are supported by the DBMS. The data definition language is just the specific set of language constructs available to describe an actual application's data in terms of the *data model*.

Exercise 1.8 Describe the structure of a DBMS. If your operating system is upgraded to support some new functions on OS files (e.g., the ability to force some sequence of bytes to disk), which layer(s) of the DBMS would you have to rewrite in order to take advantage of these new functions?

Answer 1.8 Answer omitted.

Exercise 1.9 Answer the following questions:

1. What is a transaction?
2. Why does a DBMS interleave the actions of different transactions, instead of executing transactions one after the other?
3. What must a user guarantee with respect to a transaction and database consistency? What should a DBMS guarantee with respect to concurrent execution of several transactions and database consistency?
4. Explain the strict two-phase locking protocol.
5. What is the WAL property, and why is it important?

Answer 1.9 Let us answer each question in turn:

1. A transaction is any one execution of a user program in a DBMS. This is the basic unit of change in a DBMS.

2. A DBMS is typicially shared among many users. Transactions from these users can be interleaved to improve the execution time of users' queries. By interleaving queries, users do not have to wait for other user's transactions to complete fully before their own transaction begins. Withouth interleaving, if user A begins a transaction that will take 10 seconds to complete, and user B wants to begin a transaction, user B would have to wait an additional 10 seconds for user A's transaction to complete before the database would begin processing user B's request.

3. A user must guarantee that his or her transaction does not corrupt data or insert nonsense in the database. For example, in a banking database, a user must guarantee that a cash withdraw transaction accurately models the amount a person removes from his or her account. A database application would be worthless if a person removed 20 dollars from an ATM but the transaction set their balance to zero! A DBMS must gaurantee that transactions are executed fully and independently of other transactions. An essential property of a DBMS is that a transaction should execute atomically, or as if it is the only transaction running. Also, transactions will either complete fully, or will be aborted and the database returned to it's initial state. This ensures that the database remains consistent.

4. Strict two-phase locking uses shared and exclusive locks to protect data. A transaction must hold all the required locks before executing, and does not release any lock until the transaction has completely finished.

5. The WAL property affects the logging strategy in a DBMS. The WAL, Write-Ahead Log, property states that each write action must be recorded in the log (on disk) before the corresponding change is reflected in the database itself. This protects the database from system crashes that happen during a transaction's execution. By recording the change in a log before the change is truly made, the database knows to undo the changes to recover from a system crash. Otherwise, if the system crashes just after making the change in the database but before the database logs the change, then the database would not be able to detect his change during crash recovery.

2

THE ENTITY-RELATIONSHIP MODEL

Exercise 2.1 Explain the following terms briefly: *attribute, domain, entity, relationship, entity set, relationship set, one-to-many relationship, many-to-many relationship, participation constraint, overlap constraint, covering constraint, weak entity set, aggregation*, and *role indicator*.

Answer 2.1 Term explanations:

- *Attribute* - a property or description of an entity. A toy department employee entity could have attributes describing the empoyee's name, salary, and years of service.
- *Domain* - a set of possible values for an attribute.
- *Entity* - an object inthe real world that is distinguishable from other objects such as the green dragon toy.
- *Relationship* - an association among two or more entities.
- *Entity set* - a collection of similar entities such as all of the toys in the toy department.
- *Relationship set* - a collection of similar relationships
- *One-to-many relationship* - a key constraint that indicates that one entity can be associated with many of another entity. An example of a one-to-many relationship is when an employee can work for only one department, and a department can have many employees.
- *Many-to-many relationship* - a key constraint that indicates that many of one entity can be associated with many of another entity. An example of a many-to-many relationship is employees and their hobbies: a person can have many different hobbies, and many people can have the same hobby.

- *Participation constraint* - a participation constraint determines whether relationships must involve certain entities. An example is if evry department entity has a manager entity. Participation constraints can either be total or partial. A total participation constraint says that every department has a manager. A partial participation constraint says that every employee does not have to be a manager.

- *Overlap constraint* - within an ISA hierarchy, an overlap constraint determines whether or not two subclasses can contain the same entity.

- *Covering constraint* - within an ISA hierarchy, a covering constraint determines where the entities in the subclasses collectively include all entities in the superclass. For example, with an Employees entity set with subclasses HourlyEmployee and SalaryEmployee, does every Employee entity necessarily have to be within either HourlyEmployee or SalaryEmployee?

- *Weak entity set* - is an entity that cannot be identified uniquely without considering some primary key attributes of another identifying owner entity. An example is including Dependent information for employees for insurance purposes.

- *Aggregation* - a feature of the entity relationship model that allows a relationship set to participate in another relationship set. This is indicated on an ER diagram by drawing a dashed box arround the aggregation.

- *Role indicator* - If an entity set plays more than one role, role indicators describe the different purpose in the relationship. An example is a single Employee entity set with a relation Reports-To that relates supervisors and subordinates.

Exercise 2.2 A university database contains information about professors (identified by social security number, or SSN) and courses (identified by courseid). Professors teach courses; each of the following situations concerns the Teaches relationship set. For each situation, draw an ER diagram that describes it (assuming that no further constraints hold).

1. Professors can teach the same course in several semesters, and each offering must be recorded.

2. Professors can teach the same course in several semesters, and only the most recent such offering needs to be recorded. (Assume this condition applies in all subsequent questions.)

3. Every professor must teach some course.

4. Every professor teaches exactly one course (no more, no less).

5. Every professor teaches exactly one course (no more, no less), and every course must be taught by some professor.

6. Now suppose that certain courses can be taught by a team of professors jointly, but it is possible that no one professor in a team can teach the course. Model this situation, introducing additional entity sets and relationship sets if necessary.

Answer 2.2 Answer omitted.

Exercise 2.3 Consider the following information about a university database:

- Professors have an SSN, a name, an age, a rank, and a research specialty.
- Projects have a project number, a sponsor name (e.g., NSF), a starting date, an ending date, and a budget.
- Graduate students have an SSN, a name, an age, and a degree program (e.g., M.S. or Ph.D.).
- Each project is managed by one professor (known as the project's principal investigator).
- Each project is worked on by one or more professors (known as the project's co-investigators).
- Professors can manage and/or work on multiple projects.
- Each project is worked on by one or more graduate students (known as the project's research assistants).
- When graduate students work on a project, a professor must supervise their work on the project. Graduate students can work on multiple projects, in which case they will have a (potentially different) supervisor for each one.
- Departments have a department number, a department name, and a main office.
- Departments have a professor (known as the chairman) who runs the department.
- Professors work in one or more departments, and for each department that they work in, a time percentage is associated with their job.
- Graduate students have one major department in which they are working on their degree.
- Each graduate student has another, more senior graduate student (known as a student advisor) who advises him or her on what courses to take.

Design and draw an ER diagram that captures the information about the university. Use only the basic ER model here, that is, entities, relationships, and attributes. Be sure to indicate any key and participation constraints.

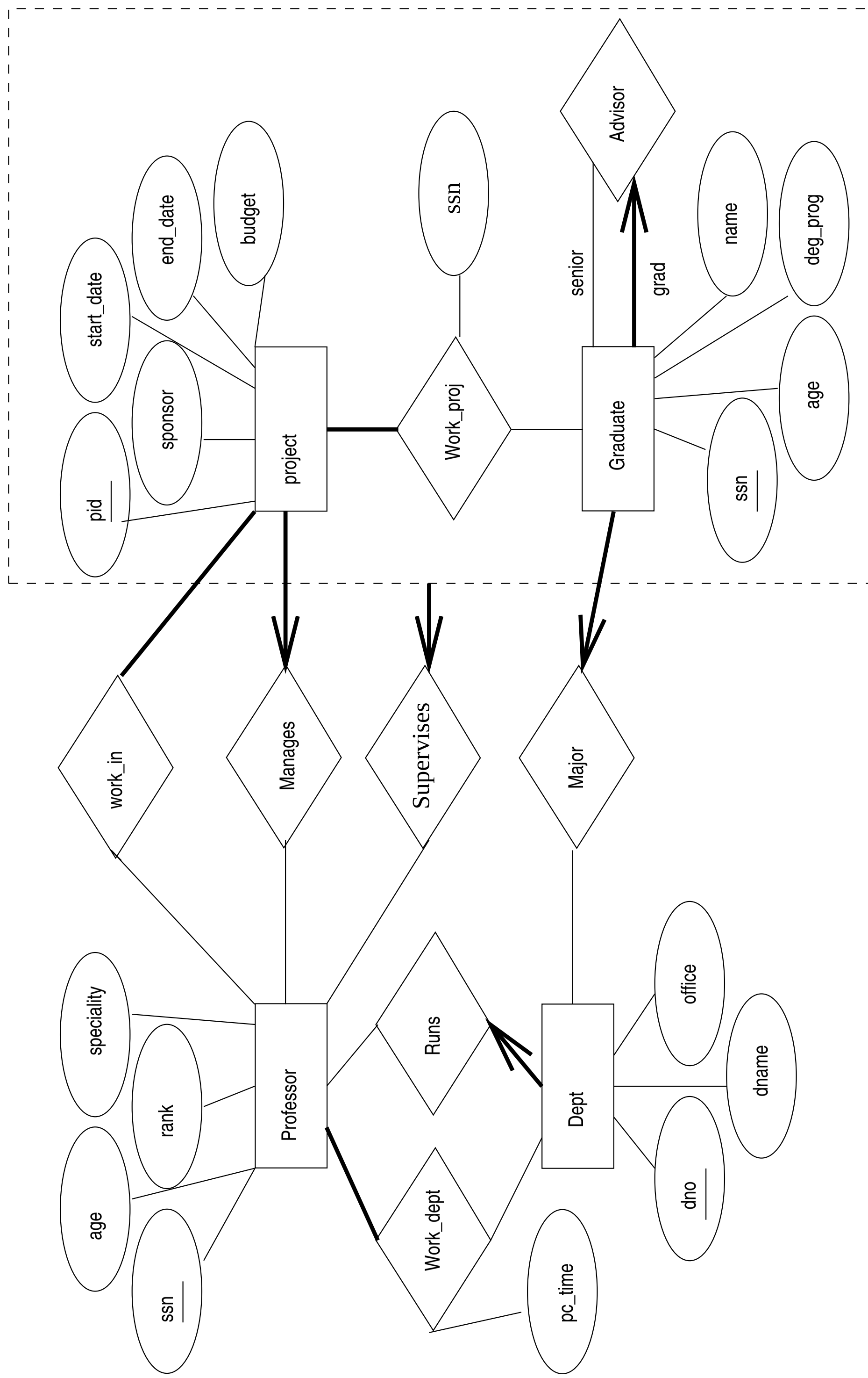

Figure 2.1 ER Diagram for Exercise 2.3

Answer 2.3 The ER diagram is shown in Figure 2.1.

Exercise 2.4 A company database needs to store information about employees (identified by *ssn*, with *salary* and *phone* as attributes); departments (identified by *dno*, with *dname* and *budget* as attributes); and children of employees (with *name* and *age* as attributes). Employees *work* in departments; each department is *managed by* an employee; a child must be identified uniquely by *name* when the parent (who is an employee; assume that only one parent works for the company) is known. We are not interested in information about a child once the parent leaves the company.

Draw an ER diagram that captures this information.

Answer 2.4 Answer omitted.

Exercise 2.5 Notown Records has decided to store information about musicians who perform on its albums (as well as other company data) in a database. The company has wisely chosen to hire you as a database designer (at your usual consulting fee of $2,500/day).

- Each musician that records at Notown has an SSN, a name, an address, and a phone number. Poorly paid musicians often share the same address, and no address has more than one phone.
- Each instrument that is used in songs recorded at Notown has a name (e.g., guitar, synthesizer, flute) and a musical key (e.g., C, B-flat, E-flat).
- Each album that is recorded on the Notown label has a title, a copyright date, a format (e.g., CD or MC), and an album identifier.
- Each song recorded at Notown has a title and an author.
- Each musician may play several instruments, and a given instrument may be played by several musicians.
- Each album has a number of songs on it, but no song may appear on more than one album.
- Each song is performed by one or more musicians, and a musician may perform a number of songs.
- Each album has exactly one musician who acts as its producer. A musician may produce several albums, of course.

Design a conceptual schema for Notown and draw an ER diagram for your schema. The above information describes the situation that the Notown database must model. Be sure to indicate all key and cardinality constraints and any assumptions that you make. Identify any constraints that you are unable to capture in the ER diagram and briefly explain why you could not express them.

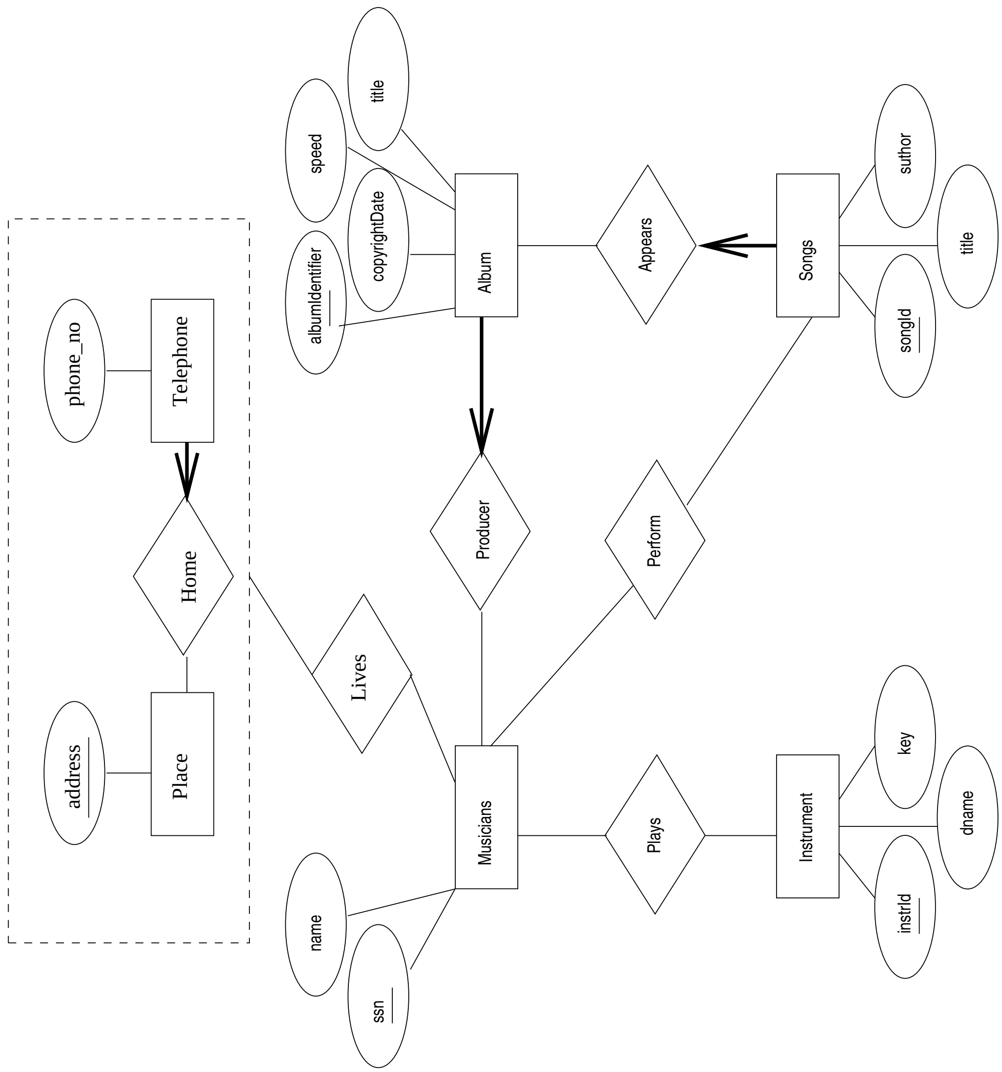

Figure 2.2 ER Diagram for Exercise 2.5

Answer 2.5 The ER diagram is shown in Figure 2.2.

Exercise 2.6 Computer Sciences Department frequent fliers have been complaining to Dane County Airport officials about the poor organization at the airport. As a result, the officials have decided that all information related to the airport should be organized using a DBMS, and you've been hired to design the database. Your first task is to organize the information about all the airplanes that are stationed and maintained at the airport. The relevant information is as follows:

- Every airplane has a registration number, and each airplane is of a specific model.
- The airport accommodates a number of airplane models, and each model is identified by a model number (e.g., DC-10) and has a capacity and a weight.
- A number of technicians work at the airport. You need to store the name, SSN, address, phone number, and salary of each technician.
- Each technician is an expert on one or more plane model(s), and his or her expertise may overlap with that of other technicians. This information about technicians must also be recorded.
- Traffic controllers must have an annual medical examination. For each traffic controller, you must store the date of the most recent exam.
- All airport employees (including technicians) belong to a union. You must store the union membership number of each employee. You can assume that each employee is uniquely identified by the social security number.
- The airport has a number of tests that are used periodically to ensure that airplanes are still airworthy. Each test has a Federal Aviation Administration (FAA) test number, a name, and a maximum possible score.
- The FAA requires the airport to keep track of each time that a given airplane is tested by a given technician using a given test. For each testing event, the information needed is the date, the number of hours the technician spent doing the test, and the score that the airplane received on the test.

1. Draw an ER diagram for the airport database. Be sure to indicate the various attributes of each entity and relationship set; also specify the key and participation constraints for each relationship set. Specify any necessary overlap and covering constraints as well (in English).
2. The FAA passes a regulation that tests on a plane must be conducted by a technician who is an expert on that model. How would you express this constraint in the ER diagram? If you cannot express it, explain briefly.

Answer 2.6 Answer omitted.

Exercise 2.7 The Prescriptions-R-X chain of pharmacies has offered to give you a free lifetime supply of medicines if you design its database. Given the rising cost of health care, you agree. Here's the information that you gather:

- Patients are identified by an SSN, and their names, addresses, and ages must be recorded.

- Doctors are identified by an SSN. For each doctor, the name, specialty, and years of experience must be recorded.
- Each pharmaceutical company is identified by name and has a phone number.
- For each drug, the trade name and formula must be recorded. Each drug is sold by a given pharmaceutical company, and the trade name identifies a drug uniquely from among the products of that company. If a pharmaceutical company is deleted, you need not keep track of its products any longer.
- Each pharmacy has a name, address, and phone number.
- Every patient has a primary physician. Every doctor has at least one patient.
- Each pharmacy sells several drugs and has a price for each. A drug could be sold at several pharmacies, and the price could vary from one pharmacy to another.
- Doctors prescribe drugs for patients. A doctor could prescribe one or more drugs for several patients, and a patient could obtain prescriptions from several doctors. Each prescription has a date and a quantity associated with it. You can assume that if a doctor prescribes the same drug for the same patient more than once, only the last such prescription needs to be stored.
- Pharmaceutical companies have long-term contracts with pharmacies. A pharmaceutical company can contract with several pharmacies, and a pharmacy can contract with several pharmaceutical companies. For each contract, you have to store a start date, an end date, and the text of the contract.
- Pharmacies appoint a supervisor for each contract. There must always be a supervisor for each contract, but the contract supervisor can change over the lifetime of the contract.

1. Draw an ER diagram that captures the above information. Identify any constraints that are not captured by the ER diagram.
2. How would your design change if each drug must be sold at a fixed price by all pharmacies?
3. How would your design change if the design requirements change as follows: If a doctor prescribes the same drug for the same patient more than once, several such prescriptions may have to be stored.

Answer 2.7 1. The ER diagram is shown in Figure 2.3.

2. If the drug is to be sold at a fixed price we can add the price attribute to the Drug entity set and eliminate the price from the Sell relationship set.

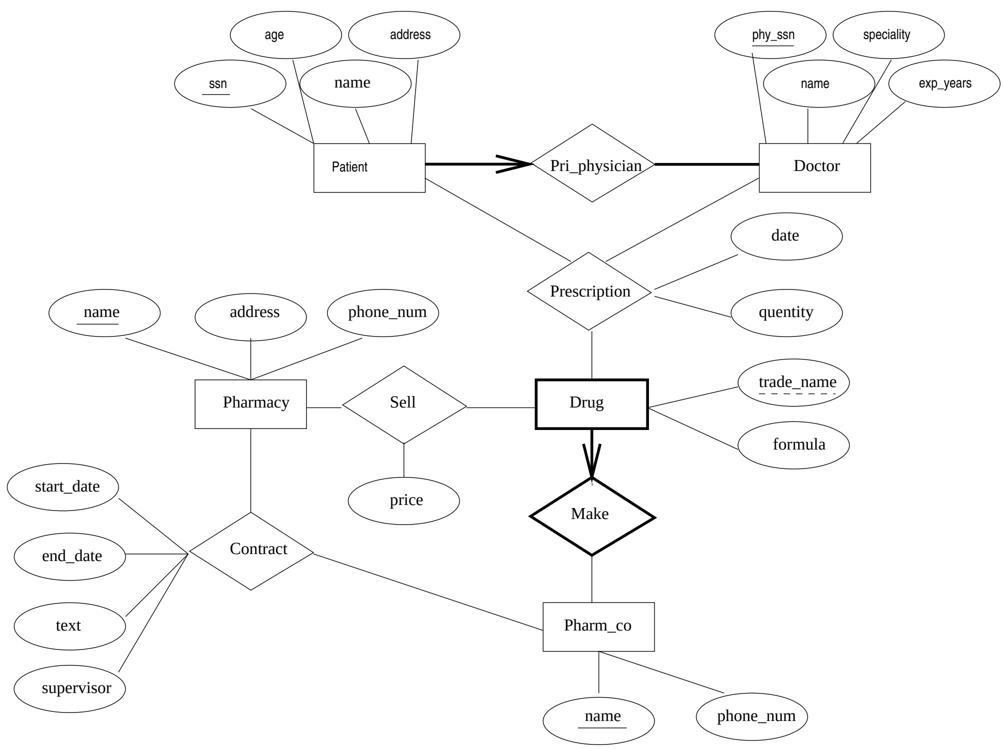

Figure 2.3 ER Diagram for Exercise 2.8

3. The date information can no longer be modeled as an attribute of Prescription. We have to create a new entity set called Prescription_date and make Prescription a 4-way relationship set that involves this additional entity set.

Exercise 2.8 Although you always wanted to be an artist, you ended up being an expert on databases because you love to cook data and you somehow confused 'data base' with 'data baste.' Your old love is still there, however, so you set up a database company, ArtBase, that builds a product for art galleries. The core of this product is a database with a schema that captures all the information that galleries need to maintain. Galleries keep information about artists, their names (which are unique), birthplaces, age, and style of art. For each piece of artwork, the artist, the year it was made, its unique title, its type of art (e.g., painting, lithograph, sculpture, photograph), and its price must be stored. Pieces of artwork are also classified into groups of various kinds, for example, portraits, still lifes, works by Picasso, or works of the 19th century; a given piece may belong to more than one group. Each group is identified by a name (like those above) that describes the group. Finally, galleries keep information about customers. For each customer, galleries keep their unique name, address, total amount of dollars they have spent in the gallery (very important!), and the artists and groups of art that each customer tends to like.

Draw the ER diagram for the database.

Answer 2.8 Answer omitted.

3

THE RELATIONAL MODEL

Exercise 3.1 Define the following terms: *relation schema, relational database schema, domain, relation instance, relation cardinality*, and *relation degree*.

Answer 3.1 A *relation schema* can be thought of as the basic information describing a table or *relation*. This includes a set of column names, the data types associated with each column, and the name associated with the entire table. For example, a relation schema for the relation called Students could be expressed using the following representation:

Students(*sid:* `string`, *name:* `string`, *login:* `string`,
age: `integer`, *gpa:* `real`)

There are five fields or columns, with names and types as shown above.

A *relational database schema* is a collection of relation schemas, describing one or more relations.

Domain is synonymous with *data type*. *Attributes* can be thought of as columns in a table. Therefore, an *attribute domain* refers to the data type associated with a column.

A *relation instance* is a set of tuples (also known as *rows* or *records*) that each conform to the schema of the relation.

The *relation cardinality* is the number of tuples in the relation.

The *relation degree* is the number of fields (or columns) in the relation.

Exercise 3.2 How many distinct tuples are in a relation instance with cardinality 22?

Answer 3.2 Answer omitted.

Exercise 3.3 Does the relational model, as seen by an SQL query writer, provide physical and logical data independence? Explain.

Answer 3.3 The user of SQL has no idea how the data is physically represented in the machine. He or she relies entirely on the relation abstraction for querying. Physical data independence is therefore assured. Since a user can define views, logical data independence can also be achieved by using view definitions to hide changes in the conceptual schema.

Exercise 3.4 What is the difference between a candidate key and the primary key for a given relation? What is a superkey?

Answer 3.4 Answer omitted.

FIELDS (ATTRIBUTES, COLUMNS)

Field names

TUPLES (RECORDS, ROWS)

sid	*name*	*login*	*age*	*gpa*
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure 3.1 An Instance $S1$ of the Students Relation

Exercise 3.5 Consider the instance of the Students relation shown in Figure 3.1.

1. Give an example of an attribute (or set of attributes) that you can deduce is *not* a candidate key, based on this instance being legal.
2. Is there any example of an attribute (or set of attributes) that you can deduce *is* a candidate key, based on this instance being legal?

Answer 3.5 Examples of non-candidate keys include the following: {name}, {age}. (Note that {gpa} can *not* be declared a non-candidate key from this evidence alone (even though common sense tells us that clearly more than one student could have the same grade point average.)

You cannot determine a key of a relation given only one instance of the relation. The fact that the instance is “legal” is immaterial. A candidate key, as defined here, *is a* *key*, not something that only *might* be a key. The instance shown is just one possible "snapshot" of the relation. At other times, the same relation may have an instance (or snapshot) that contains a totally different set of tuples, and we cannot make predictions about those instances based only upon the instance that we are given.

Exercise 3.6 What is a foreign key constraint? Why are such constraints important? What is referential integrity?

Answer 3.6 Answer omitted.

Exercise 3.7 Consider the relations Students, Faculty, Courses, Rooms, Enrolled, Teaches, and Meets_In that were defined in Section 1.5.2.

1. List all the foreign key constraints among these relations.
2. Give an example of a (plausible) constraint involving one or more of these relations that is not a primary key or foreign key constraint.

Answer 3.7 There is no reason for a foreign key constraint (FKC) on the Students, Faculty, Courses, or Rooms relations. These are the most basic relations and must be free-standing. Special care must be given to entering data into these base relations.

In the Enrolled relation, *sid* and *cid* should both have FKCs placed on them. (Real students must be enrolled in real courses.) Also, since real teachers must teach real courses, both the fid and the *cid* fields in the Teaches relation should have FKCs. Finally, Meets_In should place FKCs on both the *cid* and *rno* fields.

It would probably be wise to enforce a few other constraints on this DBMS: the length of *sid*, *cid*, and fid could be standardized; checksums could be added to these identification numbers; limits could be placed on the size of the numbers entered into the credits, capacity, and salary fields; an enumerated type should be assigned to the grade field (preventing a student from receiving a grade of G, among other things); etc.

Exercise 3.8 Answer each of the following questions briefly. The questions are based on the following relational schema:

Emp(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(<u>*eid:* `integer`, *did:* `integer`</u>, *pct_time:* `integer`)
Dept(<u>*did:* `integer`</u>, *dname:* `string`, *budget:* `real`, *managerid:* `integer`)

1. Give an example of a foreign key constraint that involves the Dept relation. What are the options for enforcing this constraint when a user attempts to delete a Dept tuple?

2. Write the SQL statements required to create the above relations, including appropriate versions of all primary and foreign key integrity constraints.

3. Define the Dept relation in SQL so that every department is guaranteed to have a manager.

4. Write an SQL statement to add 'John Doe' as an employee with $eid = 101$, $age = 32$ and $salary = 15,000$.

5. Write an SQL statement to give every employee a 10% raise.

6. Write an SQL statement to delete the 'Toy' department. Given the referential integrity constraints you chose for this schema, explain what happens when this statement is executed.

Answer 3.8 Answer omitted.

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Figure 3.2 Students with $age < 18$ on Instance $S1$

Exercise 3.9 Consider the SQL query whose answer is shown in Figure 3.2.

1. Modify this query so that only the *login* column is included in the answer.

2. If the clause WHERE *S.gpa >= 2* is added to the original query, what is the set of tuples in the answer?

Answer 3.9 The answers are as follows:

1. Only *login* is included in the answer:

```
SELECT S.login
FROM   Students S
WHERE  S.age < 18
```

2. The answer tuple for Madayan is omitted.

Exercise 3.10 Explain why the addition of NOT NULL constraints to the SQL definition of the Manages relation (in Section 3.5.3) would not enforce the constraint that each department must have a manager. What, if anything, is achieved by requiring that the *ssn* field of Manages be non-null?

Answer 3.10 Answer omitted.

Exercise 3.11 Suppose that we have a ternary relationship R between entity sets A, B, and C such that A has a key constraint and total participation and B has a key constraint; these are the only constraints. A has attributes $a1$ and $a2$, with $a1$ being the key; B and C are similar. R has no descriptive attributes. Write SQL statements that create tables corresponding to this information so as to capture as many of the constraints as possible. If you cannot capture some constraint, explain why.

Answer 3.11 The following SQL statement creates Table A:

```
CREATE TABLE A ( a1      CHAR(10),
                 a2      CHAR(10),
                 PRIMARY KEY (a1) )
```

Tables B and C are similar to A.

```
CREATE TABLE R ( a1      CHAR(10),
                 b1      CHAR(10) NOT NULL ,
                 c1      CHAR(10) ,
                 PRIMARY KEY (a1),
                 UNIQUE (b1)
                 FOREIGN KEY (a1) REFERENCES A,
                 FOREIGN KEY (b1) REFERENCES B )
                 FOREIGN KEY (c1) REFERENCES C )
```

We cannot capture the total participation constraint of A in R. This is because we cannot ensure that every key a1 appears in R without the use of checks.

Exercise 3.12 Consider the scenario from Exercise 2.2 where you designed an ER diagram for a university database. Write SQL statements to create the corresponding relations and capture as many of the constraints as possible. If you cannot capture some constraints, explain why.

Answer 3.12 Answer omitted.

Exercise 3.13 Consider the university database from Exercise 2.3 and the ER diagram that you designed. Write SQL statements to create the corresponding relations and capture as many of the constraints as possible. If you cannot capture some constraints, explain why.

Answer 3.13 The following SQL statements create the corresponding relations.

1.
```
CREATE TABLE Professors ( prof_ssn   CHAR(10),
                          name       CHAR(64),
                          age        INTEGER,
                          rank       INTEGER,
                          speciality CHAR(64),
                          PRIMARY KEY (prof_ssn) )
```

2.
```
CREATE TABLE Depts ( dno    INTEGER,
                     dname  CHAR(64),
                     office CHAR(10),
                     PRIMARY KEY (dno) )
```

3.
```
CREATE TABLE Runs ( dno      INTEGER,
                    prof_ssn CHAR(10) NOT NULL,
                    PRIMARY KEY ( dno, prof_ssn),
                    FOREIGN KEY (prof_ssn) REFERENCES Professors,
                    FOREIGN KEY (dno) REFERENCES Depts )
```

4.
```
CREATE TABLE Work_Dept ( dno      INTEGER,
                         prof_ssn CHAR(10),
                         pc_time  INTEGER,
                         PRIMARY KEY (dno, prof_ssn),
                         FOREIGN KEY (prof_ssn) REFERENCES Professors,
                         FOREIGN KEY (dno) REFERENCES Depts )
```

   Observe that we would need check constraints or assertions in SQL to enforce the rule that Professors work in at least one department.

5.
```
CREATE TABLE Project ( pid        INTEGER,
                       sponsor    CHAR(32),
                       start_date DATE,
                       end_date   DATE,
                       budget     FLOAT,
                       PRIMARY KEY (pid) )
```

6.
```
CREATE TABLE Graduates ( grad_ssn CHAR(10),
                         age      INTEGER,
                         name     CHAR(64),
                         deg_prog CHAR(32),
```

```
major INTEGER,
PRIMARY KEY (grad_ssn),
FOREIGN KEY (major) REFERENCES Depts )
```

Note that the Major table is not necessary since each Graduate has only one major and so this can be an attribute in the Graduates table.

7. ```
   CREATE TABLE Advisor ( senior_ssn CHAR(10) NOT NULL ,
                          grad_ssn CHAR(10),
                          PRIMARY KEY (senior_ssn, grad_ssn),
                          FOREIGN KEY (senior_ssn) REFERENCES Graduates,
                          FOREIGN KEY (grad_ssn) REFERENCES Graduates )
   ```

8. ```
   CREATE TABLE Manages ( pid INTEGER,
                          prof_ssn CHAR(10) NOT NULL ,
                          PRIMARY KEY (pid, prof_ssn),
                          FOREIGN KEY (prof_ssn) REFERENCES Professors,
                          FOREIGN KEY (pid) REFERENCES Projects )
   ```

9. ```
   CREATE TABLE Work_In ( pid INTEGER,
                          prof_ssn CHAR(10),
                          PRIMARY KEY (pid, prof_ssn),
                          FOREIGN KEY (prof_ssn) REFERENCES Professors,
                          FOREIGN KEY (pid) REFERENCES Projects )
   ```

   Observe that we cannot enforce the participation constraint for Projects in the Work_In table without check constraints or assertions in SQL.

10. ```
    CREATE TABLE Supervises ( prof_ssn CHAR(10) NOT NULL ,
                              grad_ssn CHAR(10),
                              pid INTEGER,
                              PRIMARY KEY (prof_ssn, grad_ssn, pid),
                              FOREIGN KEY (prof_ssn) REFERENCES Professors,
                              FOREIGN KEY (grad_ssn) REFERENCES Graduates,
                              FOREIGN KEY (pid) REFERENCES Projects )
    ```

    Note that we do not need an explicit table for the Work_Proj relation since every time a Graduate works on a Project, he or she must have a Supervisor.

Exercise 3.14 Consider the scenario from Exercise 2.4 where you designed an ER diagram for a company database. Write SQL statements to create the corresponding relations and capture as many of the constraints as possible. If you cannot capture some constraints, explain why.

Answer 3.14 Answer omitted.

Exercise 3.15 Consider the Notown database from Exercise 2.4. You have decided to recommend that Notown use a relational database system to store company data. Show the SQL statements for creating relations corresponding to the entity sets and relationship sets in your design. Identify any constraints in the ER diagram that you are unable to capture in the SQL statements and briefly explain why you could not express them.

Answer 3.15 The following SQL statements create the corresponding relations.

1.
```
CREATE TABLE Musicians ( ssn     CHAR(10),
                         name    CHAR(30),
                         PRIMARY KEY (ssn))
```

2.
```
CREATE TABLE Instruments (instrId  CHAR(10),
                          dname    CHAR(30),
                          key      CHAR(5),
                          PRIMARY KEY (instrId))
```

3.
```
CREATE TABLE Plays (     ssn      CHAR(10),
                         instrId  INTEGER,
                         PRIMARY KEY (ssn, instrId),
                         FOREIGN KEY (ssn) REFERENCES Musicians,
                         FOREIGN KEY (instrId) REFERENCES Instruments )
```

4.
```
CREATE TABLE Songs_Appears ( songId          INTEGER,
                             author          CHAR(30),
                             title           CHAR(30),
                             albumIdentifier INTEGER NOT NULL,
                             PRIMARY KEY (songId),
                             FOREIGN KEY (albumIdentifier)
                                             References Album_Producer,
```

5.
```
CREATE TABLE Telephone_Home ( phone          CHAR(11),
                              address        CHAR(30),
                              PRIMARY KEY (phone),
                              FOREIGN KEY (address) REFERENCES Place,
```

```
6. CREATE TABLE Lives (      ssn      CHAR(10),
                             phone    CHAR(11),
                             address  CHAR(30),
                             PRIMARY KEY (ssn, address),
                             FOREIGN KEY (phone, address)
                                      References Telephone_Home,
                                      FOREIGN KEY (ssn) REFERENCES Musicians )
```

```
7. CREATE TABLE Place (      address  CHAR(30) )
```

```
8. CREATE TABLE Perform (    songId   INTEGER,
                             ssn      CHAR(10),
                             PRIMARY KEY (ssn, songId),
                             FOREIGN KEY (songId) REFERENCES Songs,
                             FOREIGN KEY (ssn) REFERENCES Musicians )
```

```
9. CREATE TABLE Album_Producer ( ssn             CHAR(10) NOT NULL,
                                 albumIdentifier INTEGER,
                                 copyrightDate   DATE,
                                 speed           INTEGER,
                                 title           CHAR(30),
                                 PRIMARY KEY (albumIdentifier),
                                 FOREIGN KEY (ssn) REFERENCES Musicians )
```

Exercise 3.16 Translate your ER diagram from Exercise 2.6 into a relational schema, and show the SQL statements needed to create the relations, using only key and null constraints. If your translation cannot capture any constraints in the ER diagram, explain why.

In Exercise 2.6, you also modified the ER diagram to include the constraint that tests on a plane must be conducted by a technician who is an expert on that model. Can you modify the SQL statements defining the relations obtained by mapping the ER diagram to check this constraint?

Answer 3.16 Answer omitted.

Exercise 3.17 Consider the ER diagram that you designed for the Prescriptions-R-X chain of pharmacies in Exercise 2.7. Define relations corresponding to the entity sets and relationship sets in your design using SQL.

Answer 3.17 The statements to create tables corresponding to entity sets Doctor, Pharmacy, and Pharm_co are straightforward and omitted. The other required tables can be created as follows:

1. ```
   CREATE TABLE Pri_Phy_Patient ( ssn        CHAR(11),
                                  name       CHAR(20),
                                  age        INTEGER,
                                  address    CHAR(20),
                                  phy_ssn    CHAR(11),
                                  PRIMARY KEY (ssn),
                                  FOREIGN KEY (phy_ssn) REFERENCES Doctor )
   ```

2. ```
   CREATE TABLE Prescription ( ssn        CHAR(11),
                               phy_ssn    CHAR(11),
                               date       CHAR(11),
                               quantity   INTEGER,
                               trade_name CHAR(20),
                               pharm_id   CHAR(11),
                               PRIMARY KEY (ssn, phy_ssn),
                               FOREIGN KEY (ssn) REFERENCES Patient,
                               FOREIGN KEY (phy_ssn) REFERENCES Doctor)
                               FOREIGN KEY (trade_name, pharm_id)
                                          References Make_Drug)
   ```

3. ```
   CREATE TABLE Make_Drug (trade_name  CHAR(20),
                           pharm_id    CHAR(11),
                           PRIMARY KEY (trade_name, pharm_id),
                           FOREIGN KEY (trade_name) REFERENCES Drug,
                           FOREIGN KEY (pharm_id) REFERENCES Pharm_co)
   ```

4. ```
   CREATE TABLE Sell (     price       INTGER,
                           name        CHAR(10),
                           trade_name  CHAR(10),
                           PRIMARY KEY (name, trade_name),
                           FOREIGN KEY (name) REFERENCES Pharmacy,
                           FOREIGN KEY (trade_name) REFERENCES Drug)
   ```

5. ```
   CREATE TABLE Contract ( name        CHAR(20),
                           pharm_id    CHAR(11),
                           start_date  CHAR(11),
                           end_date    CHAR(11),
   ```
   ```

```
text          CHAR(10000),
supervisor    CHAR(20),
PRIMARY KEY (name, pharm_id),
FOREIGN KEY (name) REFERENCES Pharmacy,
FOREIGN KEY (pharm_id) REFERENCES Pharm_co)
```

Exercise 3.18 Write SQL statements to create the corresponding relations to the ER diagram you designed for Exercise 2.8. If your translation cannot capture any constraints in the ER diagram, explain why.

Answer 3.18 Answer omitted.

4

RELATIONAL ALGEBRA AND CALCULUS

Exercise 4.1 Explain the statement that relational algebra operators can be *composed*. Why is the ability to compose operators important?

Answer 4.1 Every operator in relational algebra accepts one or more relation instances as arguments and the result is always an relation instance. So the argument of one operator could be the result of another operator. This is important because, this makes it easy to write complex queries by simply composing the relational algebra operators.

Exercise 4.2 Given two relations $R1$ and $R2$, where $R1$ contains N1 tuples, $R2$ contains N2 tuples, and N2 $>$ N1 $>$ 0, give the minimum and maximum possible sizes (in tuples) for the result relation produced by each of the following relational algebra expressions. In each case, state any assumptions about the schemas for $R1$ and $R2$ that are needed to make the expression meaningful:

(1) $R1 \cup R2$, (2) $R1 \cap R2$, (3) $R1 - R2$, (4) $R1 \times R2$, (5) $\sigma_{a=5}(R1)$, (6) $\pi_a(R1)$, and (7) $R1/R2$

Answer 4.2 Answer omitted.

Exercise 4.3 Consider the following schema:

Suppliers(<u>*sid:* `integer`</u>, *sname:* `string`, *address:* `string`)
Parts(<u>*pid:* `integer`</u>, *pname:* `string`, *color:* `string`)
Catalog(<u>*sid:* `integer`, *pid:* `integer`</u>, *cost:* `real`)

The key fields are underlined, and the domain of each field is listed after the field name. Thus *sid* is the key for Suppliers, *pid* is the key for Parts, and *sid* and *pid* together form the key for Catalog. The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in relational algebra, tuple relational calculus, and domain relational calculus:

1. Find the *names* of suppliers who supply some red part.
2. Find the *sids* of suppliers who supply some red or green part.
3. Find the *sids* of suppliers who supply some red part or are at 221 Packer Ave.
4. Find the *sids* of suppliers who supply some red part and some green part.
5. Find the *sids* of suppliers who supply every part.
6. Find the *sids* of suppliers who supply every red part.
7. Find the *sids* of suppliers who supply every red or green part.
8. Find the *sids* of suppliers who supply every red part or supply every green part.
9. Find pairs of *sids* such that the supplier with the first *sid* charges more for some part than the supplier with the second *sid*.
10. Find the *pids* of parts that are supplied by at least two different suppliers.
11. Find the *pids* of the most expensive parts supplied by suppliers named Yosemite Sham.
12. Find the *pids* of parts supplied by every supplier at less than $200. (If any supplier either does not supply the part or charges more than $200 for it, the part is not selected.)

Answer 4.3 In the answers below RA refers to Relational Algebra, TRC refers to Tuple Relational Calculus and DRC refers to Domain Relational Calculus.

1. ■ RA

$$\pi_{sname}(\pi_{sid}((\pi_{pid}\sigma_{color='red'}Parts) \bowtie Catalog) \bowtie Suppliers)$$

■ TRC

$$\{T \mid \exists T1 \in Suppliers(\exists X \in Parts(X.color =' red' \wedge \exists Y \in Catalog$$
$$(Y.pid = X.pid \wedge Y.sid = T1.sid)) \wedge T.sname = T1.sname)\}$$

■ DRC

$$\{\langle Y \rangle \mid \langle X, Y, Z \rangle \in Suppliers \wedge \exists P, Q, R(\langle P, Q, R \rangle \in Parts$$
$$\wedge R =' red' \wedge \exists I, J, K(\langle I, J, K \rangle \in Catalog \wedge J = P \wedge I = X))\}$$

■ SQL

```
SELECT S.sname
FROM   Suppliers S, Parts P, Catalog C
WHERE  P.color='red' AND C.pid=P.pid AND C.sid=S.sid
```

2. ■ RA

$$\pi_{sid}(\pi_{pid}(\sigma_{color='red' \vee color='green'} Parts) \bowtie catalog)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Parts((X.color = \text{`}red' \vee X.color = \text{`}green')$$
$$\wedge X.pid = T1.pid) \wedge T.sid = T1.sid)\}$$

■ DRC

$$\{\langle X \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C \rangle \in Parts$$
$$\wedge (C =' red' \vee C =' green') \wedge A = Y)\}$$

■ SQL

```
SELECT C.sid
FROM   Catalog C, Parts P
WHERE  (P.color = 'red' OR P.color = 'green')
       AND P.pid = C.pid
```

3. ■ RA

$$\rho(R1, \pi_{sid}((\pi_{pid}\sigma_{color='red'} Parts) \bowtie Catalog))$$
$$\rho(R2, \pi_{sid}\sigma_{address='221PackerStreet'} Suppliers)$$
$$R1 \cup R2$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Parts(X.color = \text{`}red' \wedge X.pid = T1.pid)$$
$$\wedge T.sid = T1.sid)$$
$$\vee \exists T2 \in Suppliers(T2.address =' 221PackerStreet' \wedge T.sid = T2.sid)\}$$

■ DRC

$$\{\langle X \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C \rangle \in Parts$$
$$\wedge C =' red' \wedge A = Y)$$
$$\vee \exists P, Q(\langle X, P, Q \rangle \in Suppliers \wedge Q =' 221PackerStreet')\}$$

■ SQL

```
SELECT S.sid
FROM   Suppliers S
WHERE  S.address = '221 Packer street'
       OR S.sid IN ( SELECT C.sid
                     FROM   Parts P, Catalog C
                     WHERE  P.color='red' AND P.pid = C.pid )
```

4. ■ RA

$$\rho(R1, \pi_{sid}((\pi_{pid}\sigma_{color='red'}Parts) \bowtie Catalog))$$
$$\rho(R2, \pi_{sid}((\pi_{pid}\sigma_{color='green'}Parts) \bowtie Catalog))$$
$$R1 \cap R2$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Parts(X.color = `red' \wedge X.pid = T1.pid)$$
$$\wedge \exists T2 \in Catalog(\exists Y \in Parts(Y.color =' green' \wedge Y.pid = T2.pid)$$
$$\wedge T2.sid = T1.sid) \wedge T.sid = T1.sid)\}$$

■ DRC

$$\{\langle X \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C(\langle A, B, C \rangle \in Parts$$
$$\wedge C =' red' \wedge A = Y)$$
$$\wedge \exists P, Q, R(\langle P, Q, R \rangle \in Catalog \wedge \exists E, F, G(\langle E, F, G \rangle \in Parts$$
$$\wedge G =' green' \wedge E = Q) \wedge P = X)\}$$

■ SQL

```
SELECT C.sid
FROM   Parts P, Catalog C
WHERE  P.color = 'red' AND P.pid = C.pid
       AND EXISTS ( SELECT P2.pid
                    FROM   Parts P2, Catalog C2
                    WHERE  P2.color = 'green' AND C2.sid = C.sid
                           AND P2.pid = C2.pid )
```

5. ■ RA

$$(\pi_{sid,pid}Catalog)/(\pi_{pid}Parts)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\forall X \in Parts(\exists T2 \in Catalog$$
$$(T2.pid = X.pid \wedge T2.sid = T1.sid)) \wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X,Y,Z\rangle \in Catalog \land \forall\langle A,B,C\rangle \in Parts$$
$$(\exists\langle P,Q,R\rangle \in Catalog(Q = A \land P = X))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  NOT EXISTS (SELECT P.pid
                   FROM   Parts P
                   WHERE  NOT EXISTS (SELECT C1.sid
                                      FROM   Catalog C1
                                      WHERE  C1.sid = C.sid
                                             AND C1.pid = P.pid))
```

6. - RA

$$(\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='red'}Parts)$$

- TRC

$$\{T \mid \exists T1 \in Catalog(\forall X \in Parts(X.color \neq \text{`}red\text{'}$$
$$\lor \exists T2 \in Catalog(T2.pid = X.pid \land T2.sid = T1.sid))$$
$$\land T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X,Y,Z\rangle \in Catalog \land \forall\langle A,B,C\rangle \in Parts$$
$$(C \neq \text{`}red\text{'} \lor \exists\langle P,Q,R\rangle \in Catalog(Q = A \land P = X))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  NOT EXISTS (SELECT P.pid
                   FROM   Parts P
                   WHERE  P.color = 'red'
                   AND (NOT EXISTS (SELECT C1.sid
                                    FROM   Catalog C1
                                    WHERE  C1.sid = C.sid AND
                                           C1.pid = P.pid)))
```

7. - RA

$$(\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='red' \lor color='green'}Parts)$$

- TRC

$$\{T \mid \exists T1 \in Catalog(\forall X \in Parts((X.color \neq \text{`}red' \\ \wedge X.color \neq \text{`}green') \vee \exists T2 \in Catalog \\ (T2.pid = X.pid \wedge T2.sid = T1.sid)) \wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \forall \langle A, B, C\rangle \in Parts \\ ((C \neq \text{`}red' \wedge C \neq \text{`}green') \vee \exists \langle P, Q, R\rangle \in Catalog \\ (Q = A \wedge P = X))\}$$

- SQL

```
SELECT  C.sid
FROM    Catalog C
WHERE   NOT EXISTS (SELECT  P.pid
                    FROM    Parts P
                    WHERE   (P.color = 'red' OR P.color = 'green')
                    AND (NOT EXISTS (SELECT  C1.sid
                                     FROM    Catalog C1
                                     WHERE   C1.sid = C.sid AND
                                             C1.pid = P.pid)))
```

8. - RA

$$\rho(R1, ((\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='red'}Parts))) \\ \rho(R2, ((\pi_{sid,pid}Catalog)/(\pi_{pid}\sigma_{color='green'}Parts))) \\ R1 \cup R2$$

- TRC

$$\{T \mid \exists T1 \in Catalog((\forall X \in Parts \\ (X.color \neq \text{`}red' \vee \exists Y \in Catalog(Y.pid = X.pid \wedge Y.sid = T1.sid)) \\ \vee \forall Z \in Parts(Z.color \neq \text{`}green' \vee \exists P \in Catalog \\ (P.pid = Z.pid \wedge P.sid = T1.sid))) \wedge T.sid = T1.sid)\}$$

- DRC

$$\{\langle X\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge (\forall \langle A, B, C\rangle \in Parts \\ (C \neq \text{`}red' \vee \exists \langle P, Q, R\rangle \in Catalog(Q = A \wedge P = X)) \\ \vee \forall \langle U, V, W\rangle \in Parts(W \neq \text{`}green' \vee \langle M, N, L\rangle \in Catalog \\ (N = U \wedge M = X)))\}$$

- SQL

```
SELECT C.sid
FROM   Catalog C
WHERE  (NOT EXISTS (SELECT P.pid
                    FROM   Parts P
                    WHERE  P.color = 'red' AND
                    (NOT EXISTS (SELECT C1.sid
                                 FROM   Catalog C1
                                 WHERE  C1.sid = C.sid AND
                                        C1.pid = P.pid))))
       OR ( NOT EXISTS (SELECT P1.pid
                        FROM   Parts P1
                        WHERE  P1.color = 'green' AND
                        (NOT EXISTS (SELECT C2.sid
                                     FROM   Catalog C2
                                     WHERE  C2.sid = C.sid AND
                                            C2.pid = P1.pid))))
```

9. - RA

$$\rho(R1, Catalog)$$
$$\rho(R2, Catalog)$$
$$\pi_{R1.sid,R2.sid}(\sigma_{R1.pid=R2.pid \wedge R1.sid \neq R2.sid \wedge R1.cost > R2.cost}(R1 \times R2))$$

- TRC

$$\begin{aligned}\{T \quad | \quad & \exists T1 \in Catalog(\exists T2 \in Catalog \\ & (T2.pid = T1.pid \wedge T2.sid \neq T1.sid \\ & \wedge T2.cost < T1.cost \wedge T.sid2 = T2.sid) \\ & \wedge T.sid1 = T1.sid)\}\end{aligned}$$

- DRC

$$\begin{aligned}&\{\langle X, P\rangle \mid \langle X, Y, Z\rangle \in Catalog \wedge \exists P, Q, R \\ &(\langle P, Q, R\rangle \in Catalog \wedge Q = Y \wedge P \neq X \wedge R < Z)\}\end{aligned}$$

- SQL

```
SELECT C1.sid, C2.sid
FROM   Catalog C1, Catalog C2
WHERE  C1.pid = C2.pid AND C1.sid ≠ C2.sid
       AND C1.cost > C2.cost
```

10. ■ RA

$$\rho(R1, Catalog)$$
$$\rho(R2, Catalog)$$
$$\pi_{R1.pid}\sigma_{R1.pid=R2.pid \wedge R1.sid \neq R2.sid}(R1 \times R2)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists T2 \in Catalog$$
$$(T2.pid = T1.pid \wedge T2.sid \neq T1.sid)$$
$$\wedge T.pid = T1.pid)\}$$

■ DRC

$$\{\langle X \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C$$
$$(\langle A, B, C \rangle \in Catalog \wedge B = Y \wedge A \neq X)\}$$

■ SQL

```
SELECT C.pid
FROM   Catalog C
WHERE  EXISTS (SELECT C1.sid
               FROM   Catalog C1
               WHERE  C1.pid = C.pid AND C1.sid ≠ C.sid )
```

11. ■ RA

$$\rho(R1, \pi_{sid}\sigma_{sname='YosemiteSham'} Suppliers)$$
$$\rho(R2, R1 \bowtie Catalog)$$
$$\rho(R3, R2)$$
$$\rho(R4(1 \rightarrow sid, 2 \rightarrow pid, 3 \rightarrow cost), \sigma_{R3.cost < R2.cost}(R3 \times R2))$$
$$\pi_{pid}(R2 - \pi_{sid,pid,cost} R4)$$

■ TRC

$$\{T \mid \exists T1 \in Catalog(\exists X \in Suppliers$$
$$(X.sname =' YosemiteSham' \wedge X.sid = T1.sid) \wedge \neg(\exists S \in Suppliers$$
$$(S.sname =' YosemiteSham' \wedge \exists Z \in Catalog$$
$$(Z.sid = S.sid \wedge Z.cost > T1.cost))) \wedge T.pid = T1.pid)$$

■ DRC

$$\{\langle Y \rangle \mid \langle X, Y, Z \rangle \in Catalog \wedge \exists A, B, C$$
$$(\langle A, B, C \rangle \in Suppliers \wedge C =' YosemiteSham' \wedge A = X)$$
$$\wedge \neg(\exists P, Q, R(\langle P, Q, R \rangle \in Suppliers \wedge R =' YosemiteSham'$$
$$\wedge \exists I, J, K(\langle I, J, K \rangle \in Catalog(I = P \wedge K > Z))))\}$$

- SQL

```
SELECT C.pid
FROM   Catalog C, Suppliers S
WHERE  S.sname = 'Yosemite Sham' AND C.sid = S.sid
       AND C.cost ≥ ALL (Select C2.cost
                         FROM  Catalog C2, Suppliers S2
                         WHERE S2.sname = 'Yosemite Sham'
                               AND C2.sid = S2.sid)
```

Exercise 4.4 Consider the Supplier-Parts-Catalog schema from the previous question. State what the following queries compute:

1. $\pi_{sname}(\pi_{sid}(\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)$
2. $\pi_{sname}(\pi_{sid}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers))$
3. $(\pi_{sname}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)) \ \cap$

   $(\pi_{sname}((\sigma_{color='green'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers))$
4. $(\pi_{sid}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)) \ \cap$

   $(\pi_{sid}((\sigma_{color='green'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers))$
5. $\pi_{sname}((\pi_{sid,sname}((\sigma_{color='red'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)) \ \cap$

   $(\pi_{sid,sname}((\sigma_{color='green'}Parts) \bowtie (\sigma_{cost<100}Catalog) \bowtie Suppliers)))$

Answer 4.4 The statements can be interpreted as:

1. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars.
2. This Relational Algebra statement does not return anything because of the sequence of projection operators. Once the sid is projected, it is the only field in the set. Therefore, projecting on sname will nt return anything.
3. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.
4. Find the Supplier ids of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.
5. Find the Supplier names of the suppliers who supply a red part that costs less than 100 dollars and a green part that costs less than 100 dollars.

Exercise 4.5 Consider the following relations containing airline flight information:

> Flights(*flno:* `integer`, *from:* `string`, *to:* `string`,
> *distance:* `integer`, *departs:* `time`, *arrives:* `time`)
> Aircraft(*aid:* `integer`, *aname:* `string`, *cruisingrange:* `integer`)
> Certified(*eid:* `integer`, *aid:* `integer`)
> Employees(*eid:* `integer`, *ename:* `string`, *salary:* `integer`)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft (otherwise, he or she would not qualify as a pilot), and only pilots are certified to fly.

Write the following queries in relational algebra, tuple relational calculus, and domain relational calculus. Note that some of these queries may not be expressible in relational algebra (and, therefore, also not expressible in tuple and domain relational calculus)! For such queries, informally explain why they cannot be expressed. (See the exercises at the end of Chapter 5 for additional queries over the airline schema.)

1. Find the *eid*s of pilots certified for some Boeing aircraft.
2. Find the *name*s of pilots certified for some Boeing aircraft.
3. Find the *aid*s of all aircraft that can be used on non-stop flights from Bonn to Madras.
4. Identify the flights that can be piloted by every pilot whose salary is more than $100,000.
5. Find the names of pilots who can operate planes with a range greater than 3,000 miles but are not certified on any Boeing aircraft.
6. Find the *eid*s of employees who make the highest salary.
7. Find the *eid*s of employees who make the second highest salary.
8. Find the *eid*s of employees who are certified for the largest number of aircraft.
9. Find the *eid*s of employees who are certified for exactly three aircraft.
10. Find the total amount paid to employees as salaries.
11. Is there a sequence of flights from Madison to Timbuktu? Each flight in the sequence is required to depart from the city that is the destination of the previous flight; the first flight must leave Madison, the last flight must reach Timbuktu, and there is no restriction on the number of intermediate flights. Your query must determine whether a sequence of flights from Madison to Timbuktu exists for *any* input Flights relation instance.

Answer 4.5 In the answers below RA refers to Relational Algebra, TRC refers to Tuple Relational Calculus and DRC refers to Domain Relational Calculus.

1. - RA

   $$\pi_{eid}(\sigma_{aname=\text{`}Boeing'}(Aircraft \bowtie Certified))$$

   - TRC

   $$\{C.eid \mid C \in Certified \wedge \\ \exists A \in Aircraft(A.aid = C.aid \wedge A.aname = \text{`}Boeing')\}$$

   - DRC

   $$\{\langle Ceid\rangle \mid \langle Ceid, Caid\rangle \in Certified \wedge \\ \exists Aid, AN, AR(\langle Aid, AN, AR\rangle \in Aircraft \\ \wedge Aid = Caid \wedge AN = \text{`}Boeing')\}$$

   - SQL

   ```
   SELECT C.eid
   FROM   Aircraft A, Certified C
   WHERE  A.aid = C.aid AND A.aname = 'Boeing'
   ```

2. - RA

   $$\pi_{ename}(\sigma_{aname=\text{`}Boeing'}(Aircraft \bowtie Certified \bowtie Employees))$$

   - TRC

   $\{E.ename \mid E \in Employees \wedge \exists C \in Certified$
   $(\exists A \in Aircraft(A.aid = C.aid \wedge A.aname = \text{`}Boeing' \wedge E.eid = C.eid))\}$

   - DRC

   $\{\langle EN\rangle \mid \langle Eid, EN, ES\rangle \in Employess \wedge$
   $\exists Ceid, Caid(\langle Ceid, Caid\rangle \in Certified \wedge$
   $\exists Aid, AN, AR(\langle Aid, AN, AR\rangle \in Aircraft \wedge$
   $Aid = Caid \wedge AN = \text{`}Boeing' \wedge Eid = Ceid)\}$

   - SQL

   ```
   SELECT E.ename
   FROM   Aircraft A, Certified C, Employees E
   WHERE  A.aid = C.aid AND A.aname = 'Boeing' AND E.eid = C.eid
   ```

3. ■ RA

$\rho(BonnToMadrid, \sigma_{from='Bonn' \wedge to='Madrid'}(Flights))$
$\pi_{aid}(\sigma_{cruisingrange>distance}(Aircraft \times BonnToMadrid))$

■ TRC

$\{A.aid \mid A \in Aircraft \wedge \exists F \in Flights$
$(F.from = 'Bonn' \wedge F.to = 'Madrid' \wedge A.cruisingrange > F.distance)\}$

■ DRC

$\{Aid \mid \langle Aid, AN, AR\rangle \in Aircraft \wedge$
$(\exists FN, FF, FT, FDi, FDe, FA(\langle FN, FF, FT, FDi, FDe, FA\rangle \in Flights \wedge$
$FF = 'Bonn' \wedge FT = 'Madrid' \wedge FDi < AR))\}$

■ SQL

```
SELECT A.aid
FROM   Aircraft A, Flights F
WHERE  F.from = 'Bonn' AND F.to = 'Madrid' AND
       A.cruisingrange > F.distance
```

4. ■ RA

$\pi_{flno}(\sigma_{distance<cruisingrange \wedge salary>100,000}(Flights \bowtie Aircraft \bowtie$
$Certified \bowtie Employees)))$

■ TRC $\{F.flno \mid F \in Flights \wedge \exists A \in Aircraft \exists C \in Certified$
$\exists E \in Employees(A.cruisingrange > F.distance \wedge E.salary > 100,000 \wedge$
$A.aid = C.aid \wedge E.eid = C.eid)\}$

■ DRC

$\{FN \mid \langle FN, FF, FT, FDi, FDe, FA\rangle \in Flights \wedge$
$\exists Ceid, Caid(\langle Ceid, Caid\rangle \in Certified \wedge$
$\exists Aid, AN, AR(\langle Aid, AN, AR\rangle \in Aircraft \wedge$
$\exists Eid, EN, ES(\langle Eid, EN, ES\rangle \in Employees$
$(AR > FDi \wedge ES > 100,000 \wedge Aid = Caid \wedge Eid = Ceid)\}$

■ SQL

```
SELECT E.ename
FROM   Aircraft A, Certified C, Employees E, Flights F
WHERE  A.aid = C.aid AND E.eid = C.eid AND
       distance < cruisingrange AND salary > 100,000
```

5. ■ RA $\rho(R1, \pi_{eid}(\sigma_{cruisingrange>3000}(Aircraft \bowtie Certified)))$
$\pi_{ename}(Employees \bowtie (R1 - \pi_{eid}(\sigma_{aname=\text{'}Boeing'}(Aircraft \bowtie Certified))))$

■ TRC
$\{E.ename \mid E \in Employees \wedge \exists C \in Certified(\exists A \in Aircraft$
$(A.aid = C.aid \wedge E.eid = C.eid \wedge A.cruisingrange > 3000)) \wedge$
$\neg(\exists C2 \in Certified(\exists A2 \in Aircraft(A2.aname = \text{'}Boeing' \wedge C2.aid =$
$A2.aid \wedge C2.eid = E.eid)))\}$

■ DRC
$\{\langle EN \rangle \mid \langle Eid, EN, ES \rangle \in Employess \wedge$
$\exists Ceid, Caid(\langle Ceid, Caid \rangle \in Certified \wedge$
$\exists Aid, AN, AR(\langle Aid, AN, AR \rangle \in Aircraft \wedge$
$Aid = Caid \wedge Eid = Ceid \wedge AR > 3000)) \wedge$
$\neg(\exists Aid2, AN2, AR2(\langle Aid2, AN2, AR2 \rangle \in Aircraft \wedge$
$\exists Ceid2, Caid2(\langle Ceid2, Caid2 \rangle \in Certified$
$\wedge Aid2 = Caid2 \wedge Eid = Ceid2 \wedge AN2 = \text{'}Boeing')))\}$

■ SQL

```
SELECT E.ename
FROM   Certified C, Employees E, Aircraft A
WHERE  A.aid = C.aid AND E.eid = C.eid AND A.cruisingrange > 3000
AND E.eid NOT IN ( SELECT C2.eid
FROM Certified C2, Aircraft A2
WHERE C2.aid = A2.aid AND A2.aname = 'Boeing' )
```

6. ■ RA
The approach to take is first find all the employees who do not have the highest salary. Subtract these from the original list of employees and what is left is the highest paid employees.
$\rho(E1, Employees)$
$\rho(E2, Employees)$
$\rho(E3, \pi_{E2.eid}(E1 \bowtie_{E1.salary>E2.salary} E2)$
$(\pi_{eid}E1) - E3$

■ TRC

$\{E1.eid \mid E1 \in Employees \wedge \neg(\exists E2 \in Employees(E2.salary > E1.salary))\}$

■ DRC

$\{\langle Eid1\rangle \mid \langle Eid1, EN1, ES1\rangle \in Employess \wedge$
$\neg(\exists Eid2, EN2, ES2(\langle Eid2, EN2, ES2\rangle \in Employess \wedge ES2 > ES1))\}$

- SQL

```
SELECT  E.eid
FROM    Employees E
WHERE   E.salary = ( Select MAX (E2.salary)
                     FROM   Employees E2 )
```

7. - RA

The approach taken is similar to the solution for the previous exercise. First find all the employees who do not have the highest salary. Remove these from the original list of employees and what is left is the highest paid employees. Remove the highest paid employees from the original list. What is left is the second highest paid employees together with the rest of the employees. Then find the highest paid employees of this new list. This is the list of the second highest paid employees.
$\rho(E1, Employees)$
$\rho(E2, Employees)$
$\rho(E3, \pi_{E2.eid}(E1 \bowtie_{E1.salary>E2.salary} E2)$
$\rho(E4, E2 \bowtie E3)$
$\rho(E5, E2 \bowtie E3)$
$\rho(E6, \pi_{E5.eid}(E4 \bowtie_{E1.salary>E5.salary} E5)$
$(\pi_{eid}E3) - E6$

- TRC

$\{E1.eid \mid E1 \in Employees \wedge \exists E2 \in Employees(E2.salary > E1.salary$
$\wedge\neg(\exists E3 \in Employees(E3.salary > E2.salary)))\}$

- DRC

$\{\langle Eid1\rangle \mid \langle Eid1, EN1, ES1\rangle \in Employess \wedge$
$\exists Eid2, EN2, ES2(\langle Eid2, EN2, ES2\rangle \in Employess(ES2 > ES1)$
$\wedge\neg(\exists Eid3, EN3, ES3(\langle Eid3, EN3, ES3\rangle \in Employess(ES3 > ES2))))\}$

- SQL

```
SELECT  E.eid
FROM    Employees E
WHERE   E.salary = (SELECT MAX (E2.salary)
                    FROM   Employees E2
                    WHERE  E2.salary ≠ (SELECT MAX (E3.salary)
                                        FROM   Employees E3 ))
```

8. This cannot be expressed in relational algebra (or calculus) because there is no operator to count, and this query requires the ability to count up to a number that depends on the data. The query can however be expressed in SQL as follows:

```
SELECT  Temp.eid
FROM    ( SELECT    C.eid AS eid, COUNT (C.aid) AS cnt,
          FROM      Certified C
          GROUP BY  C.eid) AS Temp
WHERE   Temp.cnt = ( SELECT   MAX (Temp.cnt)
                     FROM     Temp)
```

9. ■ RA

   The approach behind this query is to first find the employees who are certified for at least three aircraft (they appear at least three times in the Certified relation). Then find the employees who are certified for at least four aircraft. Subtract the second from the first and what is left is the employees who are certified for exactly three aircraft.

   $\rho(R1, Certified)$
   $\rho(R2, Certified)$
   $\rho(R3, Certified)$
   $\rho(R4, Certified)$
   $\rho(R5, \pi_{eid}(\sigma_{(R1.eid=R2.eid=R3.eid)\wedge(R1.aid\neq R2.aid\neq R3.aid)}(R1 \times R2 \times R3)))$
   $\rho(R6, \pi_{eid}(\sigma_{(R1.eid=R2.eid=R3.eid=R4.eid)\wedge(R1.aid\neq R2.aid\neq R3.aid\neq R4.aid)}$
   $(R1 \times R2 \times R3 \times R4)))$
   $R5 - R6$

   ■ TRC

   $\{C1.eid \mid C1 \in Certified \wedge \exists C2 \in Certified(\exists C3 \in Certified$
   $(C1.eid = C2.eid \wedge C2.eid = C3.eid \wedge$
   $C1.aid \neq C2.aid \wedge C2.aid \neq C3.aid \wedge C3.aid \neq C1.aid \wedge$
   $\neg(\exists C4 \in Certified$
   $(C3.eid = C4.eid \wedge C1.aid \neq C4.aid \wedge$
   $C2.aid \neq C4.aid \wedge C3.aid \neq C4.aid))))\}$

   ■ DRC

   $\{\langle CE1\rangle \mid \langle CE1, CA1\rangle \in Certified \wedge$
   $\exists CE2, CA2(\langle CE2, CA2\rangle \in Certified \wedge$
   $\exists CE3, CA3(\langle CE3, CA3\rangle \in Certified \wedge$
   $(CE1 = CE2 \wedge CE2 = CE3 \wedge$
   $CA1 \neq CA2 \wedge CA2 \neq CA3 \wedge CA3 \neq CA1 \wedge$
   $\neg(\exists CE4, CA4(\langle CE4, CA4\rangle \in Certified \wedge$

$(CE3 = CE4 \wedge CA1 \neq CA4 \wedge$
$CA2 \neq CA4 \wedge CA3 \neq CA4))))\}$

- SQL

```
SELECT C1.eid
FROM   Certified C1, Certified C2, Certified C3
WHERE  (C1.eid = C2.eid AND C2.eid = C3.eid AND
       C1.aid ≠ C2.aid AND C2.aid ≠ C3.aid AND C3.aid ≠ C1.aid)
EXCEPT
SELECT C4.eid
FROM   Certified C4, Certified C5, Certified C6, Certified C7,
WHERE  (C4.eid = C5.eid AND C5.eid = C6.eid AND C6.eid = C7.eid AND
       C4.aid ≠ C5.aid AND C4.aid ≠ C6.aid AND C4.aid ≠ C7.aid AND
       C5.aid ≠ C6.aid AND C5.aid ≠ C7.aid AND C6.aid ≠ C7.aid )
```

This could also be done in SQL using COUNT.

10. This cannot be expressed in relational algebra (or calculus) because there is no operator to sum values. The query can however be expressed in SQL as follows:

```
SELECT SUM (E.salaries)
FROM   Employees E
```

11. This cannot be expressed in relational algebra or relational calculus or SQL. The problem is that there is no restriction on the number of intermediate flights. All of the query methods could find if there was a flight directly from Madison to Timbuktu and if there was a sequence of two flights that started in Madison and ended in Timbuktu. They could even find a sequence of n flights that started in Madison and ended in Timbuktu as long as there is a static (i.e., data-independent) upper bound on the number of intermediate flights. (For large n, this would of course be long and impractical, but at least possible.) In this query, however, the upper bound is not static but dynamic (based upon the set of tuples in the Flights relation).

    In summary, if we had a static upper bound (say k), we could write an algebra or SQL query that repeatedly computes (upto k) joins on the Flights relation. If the upper bound is dynamic, then we cannot write such a query because k is not known when writing the query.

Exercise 4.6 What is *relational completeness*? If a query language is relationally complete, can you write any desired query in that language?

Answer 4.6 Answer omitted.

Exercise 4.7 What is an *unsafe* query? Give an example and explain why it is important to disallow such queries.

Answer 4.7 An $unsafe$ query is a query in relational calculus that has an infinite number of results. An example of such a query is:

$$\{S \mid \neg(S \in Sailors)\}$$

The query is for all things that are not sailors which of course is everything else. Clearly there is an infinite number of answers, and this query is $unsafe$. It is important to disallow $unsafe$ queries because we want to be able to get back to users with a list of all the answers to a query after a finite amount of time.

5

SQL: QUERIES, PROGRAMMING, TRIGGERS

Exercise 5.1 Consider the following relations:

Student(*snum:* `integer`, *sname:* `string`, *major:* `string`, *level:* `string`, *age:* `integer`)
Class(*name:* `string`, *meets_at:* `time`, *room:* `string`, *fid:* `integer`)
Enrolled(*snum:* `integer`, *cname:* `string`)
Faculty(*fid*: `integer`, *fname:* `string`, *deptid:* `integer`)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

Write the following queries in SQL. No duplicates should be printed in any of the answers.

1. Find the names of all Juniors (Level = JR) who are enrolled in a class taught by I. Teach.
2. Find the age of the oldest student who is either a History major or is enrolled in a course taught by I. Teach.
3. Find the names of all classes that either meet in room R128 or have five or more students enrolled.
4. Find the names of all students who are enrolled in two classes that meet at the same time.
5. Find the names of faculty members who teach in every room in which some class is taught.
6. Find the names of faculty members for whom the combined enrollment of the courses that they teach is less than five.
7. Print the Level and the average age of students for that Level, for each Level.

8. Print the Level and the average age of students for that Level, for all Levels except JR.
9. Find the names of students who are enrolled in the maximum number of classes.
10. Find the names of students who are not enrolled in any class.
11. For each age value that appears in Students, find the level value that appears most often. For example, if there are more FR level students aged 18 than SR, JR, or SO students aged 18, you should print the pair (18, FR).

Answer 5.1 The answers are given below:

1.
```
SELECT DISTINCT S.Sname
FROM   Student S, Class C, Enrolled E, Faculty F
WHERE  S.snum = E.snum AND E.cname = C.name AND C.fid = F.fid AND
       F.fname = 'I.Teach' AND S.level = 'JR'
```

2.
```
SELECT MAX(S.age)
FROM   Student S
WHERE  (S.major = 'History')
       OR S.num IN (SELECT E.snum
                    FROM   Class C, Enrolled E, Faculty F
                    WHERE  E.cname = C.name AND C.fid = F.fid
                           AND F.fname = 'I.Teach' )
```

3.
```
SELECT   C.name
FROM     Class C
WHERE    C.room = 'R128'
         OR C.name IN (SELECT   E.cname
                       FROM     Enrolled E
                       GROUP BY E.cname
                       HAVING   COUNT (*) >= 5)
```

4.
```
SELECT DISTINCT S.sname
FROM   Student S
WHERE  S.snum IN (SELECT E1.snum
                  FROM   Enrolled E1, Enrolled E2, Class C1, Class C2
                  WHERE  E1.snum = E2.snum AND E1.cname <> E2.cname
                  AND E1.cname = C1.name
                  AND E2.cname = C2.name AND C1.time = C2.time)
```

5.
```
SELECT DISTINCT F.fname
FROM   Faculty F
WHERE  NOT EXISTS (( SELECT *
```

```
                          FROM    Class C )
                          EXCEPT
                          (SELECT C1.room
                          FROM    Class C1
                          WHERE   C1.fid = F.fid ))
```

6.
```
SELECT   DISTINCT F.fname
FROM     Faculty F
WHERE    5 > (SELECT  E.snum
              FROM    Class C, Enrolled E
              WHERE   C.name = E.cname
              AND     C.fid = F.fid)
```

7.
```
SELECT   S.level, AVG(S.age)
FROM     Student S
GROUP BY S.level
```

8.
```
SELECT   S.level, AVG(S.age)
FROM     Student S
WHERE    S.level <> 'JR'
GROUP BY S.level
```

9.
```
SELECT   DISTINCT S.sname
FROM     Student S
WHERE    S.snum IN (SELECT   E.snum
                    FROM     Enrolled E
                    GROUP BY E.snum
                    HAVING   COUNT (*) >= ALL (SELECT   COUNT (*)
                                               FROM     Enrolled E2
                                               GROUP BY E2.snum ))
```

10.
```
SELECT DISTINCT S.sname
FROM   Student S
WHERE  S.snum NOT IN (SELECT E.snum
                      FROM   Enrolled E )
```

11.
```
SELECT   S.age, S.level
FROM     Student S
GROUP BY S.age, S.level,
HAVING   S.level IN (SELECT   S1.level
                     FROM     Student S1
                     WHERE    S1.age = S.age
                     GROUP BY S1.level, S1.age
                     HAVING   COUNT (*) >= ALL (SELECT   COUNT (*)
                                                FROM     Student S2
```

```
WHERE s1.age = S2.age
GROUP BY S2.level, S2.age))
```

Exercise 5.2 Consider the following schema:

> Suppliers(*sid:* `integer`, *sname:* `string`, *address:* `string`)
> Parts(*pid:* `integer`, *pname:* `string`, *color:* `string`)
> Catalog(*sid:* `integer`, *pid:* `integer`, *cost:* `real`)

The Catalog relation lists the prices charged for parts by Suppliers. Write the following queries in SQL:

1. Find the *pname*s of parts for which there is some supplier.
2. Find the *sname*s of suppliers who supply every part.
3. Find the *sname*s of suppliers who supply every red part.
4. Find the *pname*s of parts supplied by Acme Widget Suppliers and by no one else.
5. Find the *sid*s of suppliers who charge more for some part than the average cost of that part (averaged over all the suppliers who supply that part).
6. For each part, find the *sname* of the supplier who charges the most for that part.
7. Find the *sid*s of suppliers who supply only red parts.
8. Find the *sid*s of suppliers who supply a red part and a green part.
9. Find the *sid*s of suppliers who supply a red part or a green part.

Answer 5.2 Answer omitted.

Exercise 5.3 The following relations keep track of airline flight information:

> Flights(*flno:* `integer`, *from:* `string`, *to:* `string`, *distance:* `integer`,
> *departs:* `time`, *arrives:* `time`, *price:* `integer`)
> Aircraft(*aid:* `integer`, *aname:* `string`, *cruisingrange:* `integer`)
> Certified(*eid:* `integer`, *aid:* `integer`)
> Employees(*eid:* `integer`, *ename:* `string`, *salary:* `integer`)

Note that the Employees relation describes pilots and other kinds of employees as well; every pilot is certified for some aircraft, and only pilots are certified to fly. Write each of the following queries in SQL. (*Additional queries using the same schema are listed in the exercises for Chapter 4.*)

1. Find the names of aircraft such that all pilots certified to operate them earn more than 80,000.
2. For each pilot who is certified for more than three aircraft, find the *eid* and the maximum *cruisingrange* of the aircraft that he (or she) is certified for.
3. Find the names of pilots whose *salary* is less than the price of the cheapest route from Los Angeles to Honolulu.
4. For all aircraft with *cruisingrange* over 1,000 miles, find the name of the aircraft and the average salary of all pilots certified for this aircraft.
5. Find the names of pilots certified for some Boeing aircraft.
6. Find the *aid*s of all aircraft that can be used on routes from Los Angeles to Chicago.
7. Identify the routes that can be piloted by every pilot who makes more than $100,000.
8. Print the *ename*s of pilots who can operate planes with *cruisingrange* greater than 3,000 miles, but are not certified on any Boeing aircraft.
9. A customer wants to travel from Madison to New York with no more than two changes of flight. List the choice of departure times from Madison if the customer wants to arrive in New York by 6 p.m.
10. Compute the difference between the average salary of a pilot and the average salary of all employees (including pilots).
11. Print the name and salary of every nonpilot whose salary is more than the average salary for pilots.

Answer 5.3 The answers are given below:

1.

```
SELECT DISTINCT A.aname
FROM   Aircraft A
WHERE  A.Aid IN (SELECT C.aid
                 FROM   Certified C, Employees E
                 WHERE  C.eid = E.eid AND
                 NOT EXISTS ( SELECT *
                              FROM   Employees E1
                              WHERE  E1.eid = E.eid AND E1.salary < 80000 ))
```

2.

```
SELECT   C.eid, MAX (A.cruisingrange)
FROM     Certified C, Aircraft A
WHERE    C.aid = A.aid
```

```
GROUP BY C.eid
HAVING   COUNT (*) > 3
```

3.

```
SELECT DISTINCT E.aname
FROM   Employee E
WHERE  E.salary < ( SELECT MIN (F.price)
                    FROM   Flights F
                    WHERE  F.from = 'LA' AND F.to = 'Honolulu' )
```

4. Observe that *aid* is the key for Aircraft, but the question asks for aircraft names; we deal with this complication by using an intermediate relation Temp:

```
SELECT Temp.name, Temp.AvgSalary
FROM   ( SELECT    A.aid, A.aname AS name,
                   AVG (E.salary) AS AvgSalary
         FROM      Aircraft A, Certified C, Employes E
         WHERE     A.aid = C.aid AND
                   C.eid = E.eid AND A.cruisingrange > 1000
         GROUP BY A.aid, A.aname ) AS Temp
```

5.

```
SELECT DISTINCT E.ename
FROM   Employees E, Certified C, Aircraft A
WHERE  E.eid = C.eid AND
       C.aid = A.aid AND
       A.aname = 'Boeing'
```

6.

```
SELECT A.aid
FROM   Aircraft A
WHERE  A.cruisingrange > ( SELECT MIN (F.distance)
                           FROM   Flights F
                           WHERE  F.from = 'L.A.' AND F.to = 'Chicago' )
```

7.

```
SELECT DISTINCT F.from, F.to
FROM   Flights F
WHERE  NOT EXISTS ( SELECT *
                    FROM   Employees E
                    WHERE  E.salary > 100000
                    AND
                    NOT EXISTS (SELECT *
                                FROM   Aircraft A, Certified C
                                WHERE  A.cruisingrange > F.distance
```

```
                                        AND E.eid = C.eid
                                        AND A.eid = C.aid) )
```

8.

```
SELECT DISTINCT E.ename
FROM   Employees E, Certified C, Aircraft A
WHERE  C.eid = E.eid
AND    C.aid = A.aid
AND    A.cruisingrange > 3000
AND    E.eid NOT IN ( SELECT C1.eid
       FROM Certified C1, Aircraft A1
       WHERE C1.aid = A1.aid
       AND A1.aname = 'Boeing' )
```

9.

```
SELECT F.departs
FROM   Flights F
WHERE  F.flno IN ( ( SELECT F0.flno
                     FROM   Flights F0
                     WHERE  F0.from = 'Madison' AND F0.to = 'NY' AND
                            AND F0.arrives < 1800 )
                   UNION
                   ( SELECT F0.flno
                     FROM   Flights F0, Flights F1
                     WHERE  F0.from = 'Madison' AND F0.to <> 'NY' AND
                     AND    F0.to = F1.from AND F1.to = 'NY'
                            F1.departs > F0.arrives AND
                            F1.arrives < 1800 )
                     UNION
                     (      SELECT F0.flno
                            FROM Flights F0, Flights F1, Flights F2
                            WHERE F0.from = 'Madison'
                            WHERE F0.to = F1.from
                            AND F1.to = F2.from
                            AND F2.to = 'NY'
                            AND F0.to <> 'NY'
                            AND F1.to <> 'NY'
                            AND F1.departs > F0.arrives
                            AND F2.departs > F1.arrives
                            AND F2.arrives < 1800 ))
```

10.

```
SELECT Temp1.avg - Temp2.avg
FROM   (SELECT AVG (E.salary) AS avg
        FROM   Employees E
```

```
        WHERE  E.eid IN (SELECT DISTINCT C.eid
                         FROM Certified C )) AS Temp1,
       (SELECT AVG (E1.salary) AS avg
        FROM   Employees E1 ) AS Temp2
```

11.
```
SELECT E.ename, E.salary
FROM   Employees E
WHERE  E.eid NOT IN ( SELECT DISTINCT C.eid
                      FROM   Certified C )
AND E.salary > ( SELECT AVG (E1.salary)
                 FROM   Employees E1
                 WHERE  E1.eid IN
                        ( SELECT DISTINCT C1.eid
                          FROM   Certified C1 ) )
```

Exercise 5.4 Consider the following relational schema. An employee can work in more than one department; the *pct_time* field of the Works relation shows the percentage of time that a given employee works in a given department.

Emp(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(<u>*eid:* `integer`, *did:* `integer`</u>, *pct_time:* `integer`)
Dept(<u>*did:* `integer`</u>, *budget:* `real`, *managerid:* `integer`)

Write the following queries in SQL:

1. Print the names and ages of each employee who works in both the Hardware department and the Software department.
2. For each department with more than 20 full-time-equivalent employees (i.e., where the part-time and full-time employees add up to at least that many full-time employees), print the *did* together with the number of employees that work in that department.
3. Print the name of each employee whose salary exceeds the budget of all of the departments that he or she works in.
4. Find the *managerid*s of managers who manage only departments with budgets greater than $1,000,000.
5. Find the *ename*s of managers who manage the departments with the largest budget.

sid	*sname*	*rating*	*age*
18	jones	3	30.0
41	jonah	6	56.0
22	ahab	7	44.0
63	moby	*null*	15.0

Figure 5.1 An Instance of Sailors

6. If a manager manages more than one department, he or she *controls* the sum of all the budgets for those departments. Find the *managerid*s of managers who control more than $5,000,000.

7. Find the *managerid*s of managers who control the largest amount.

Answer 5.4 Answer omitted.

Exercise 5.5 Consider the instance of the Sailors relation shown in Figure 5.1.

1. Write SQL queries to compute the average rating, using `AVG`; the sum of the ratings, using `SUM`; and the number of ratings, using `COUNT`.

2. If you divide the sum computed above by the count, would the result be the same as the average? How would your answer change if the above steps were carried out with respect to the *age* field instead of *rating*?

3. Consider the following query: *Find the names of sailors with a higher rating than all sailors with age* < 21. The following two SQL queries attempt to obtain the answer to this question. Do they both compute the result? If not, explain why. Under what conditions would they compute the same result?

```
SELECT S.sname
FROM   Sailors S
WHERE  NOT EXISTS ( SELECT *
                    FROM   Sailors S2
                    WHERE  S2.age < 21
                           AND S.rating <= S2.rating )

SELECT *
FROM   Sailors S
WHERE  S.rating > ANY ( SELECT S2.rating
                        FROM   Sailors S2
                        WHERE  S2.age < 21 )
```

4. Consider the instance of Sailors shown in Figure 5.1. Let us define instance S1 of Sailors to consist of the first two tuples, instance S2 to be the last two tuples, and S to be the given instance.
   (a) Show the left outer join of S with itself, with the join condition being *sid=sid*.
   (b) Show the right outer join of S with itself, with the join condition being *sid=sid*.
   (c) Show the full outer join of S with itself, with the join condition being *sid=sid*.
   (d) Show the left outer join of S1 with S2, with the join condition being *sid=sid*.
   (e) Show the right outer join of S1 with S2, with the join condition being *sid=sid*.
   (f) Show the full outer join of S1 with S2, with the join condition being *sid=sid*.

Answer 5.5 The answers are shown below:

1.
```
SELECT AVG (S.rating) AS AVERAGE
FROM   Sailors S

SELECT SUM (S.rating)
FROM   Sailors S

SELECT COUNT (S.rating)
FROM   Sailors S
```

2. The result using SUM and COUNT would be smaller than the result using AVERAGE if there are tuples with rating = NULL. This is because all the aggregate operators, except for COUNT, ignore NULL values. So the first approach would compute the average over all tuples while the second approach would compute the average over all tuples with non-NULL rating values. However, if the aggregation is done on the age field, the answers using both approaches would be the same since the age field does not take NULL values.
3. Only the first query is correct. The second query returns the names of sailors with a higher rating than *at least one* sailor with age $<$ 21. Note that the answer to the second query does not necessarily contain the answer to the first query. In particular, if all the sailors are at least 21 years old, the second query will return an empty set while the first query will return all the sailors. This is because the NOT EXISTS predicate in the first query will evaluate to *true* if its subquery evaluates to an empty set, while the ANY predicate in the second query will evaluate to *false* if its subquery evaluates to an empty set. The two queries give the same results if and only if one of the following two conditions hold:

   - The *Sailors* relation is empty, or

4. (a)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	*null*	15.0	63	moby	*null*	15.0

(b)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	*null*	15.0	63	moby	*null*	15.0

(c)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	18	jones	3	30.0
41	jonah	6	56.0	41	jonah	6	56.0
22	ahab	7	44.0	22	ahab	7	44.0
63	moby	*null*	15.0	63	moby	*null*	15.0

(d)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	*null*	*null*	*null*	*null*
41	jonah	6	56.0	*null*	*null*	*null*	*null*

(e)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
null	*null*	*null*	*null*	22	ahab	7	44.0
null	*null*	*null*	*null*	63	moby	*null*	15.0

(f)

sid	*sname*	*rating*	*age*	*sid*	*sname*	*rating*	*age*
18	jones	3	30.0	*null*	*null*	*null*	*null*
41	jonah	6	56.0	*null*	*null*	*null*	*null*
null	*null*	*null*	*null*	22	ahab	7	44.0
null	*null*	*null*	*null*	63	moby	*null*	15.0

- There is at least one sailor with age $>$ 21 in the *Sailors* relation, and for every sailor s, either s has a higher rating than all sailors under 21 or s has a rating no higher than all sailors under 21.

Exercise 5.6 Answer the following questions.

1. Explain the term *impedance mismatch* in the context of embedding SQL commands in a host language such as C.
2. How can the value of a host language variable be passed to an embedded SQL command?
3. Explain the `WHENEVER` command's use in error and exception handling.
4. Explain the need for cursors.
5. Give an example of a situation that calls for the use of embedded SQL, that is, interactive use of SQL commands is not enough, and some host language capabilities are needed.
6. Write a C program with embedded SQL commands to address your example in the previous answer.
7. Write a C program with embedded SQL commands to find the standard deviation of sailors' ages.
8. Extend the previous program to find all sailors whose age is within one standard deviation of the average age of all sailors.
9. Explain how you would write a C program to compute the transitive closure of a graph, represented as an SQL relation Edges(*from, to*), using embedded SQL commands. (You don't have to write the program; just explain the main points to be dealt with.)
10. Explain the following terms with respect to cursors: *updatability, sensitivity*, and *scrollability.*
11. Define a cursor on the Sailors relation that is updatable, scrollable, and returns answers sorted by *age.* Which fields of Sailors can such a cursor *not* update? Why?

12. Give an example of a situation that calls for dynamic SQL, that is, even embedded SQL is not sufficient.

Answer 5.6 Answer omitted.

Exercise 5.7 Consider the following relational schema and briefly answer the questions that follow:

> Emp(*eid:* `integer`, *ename:* `string`, *age:* `integer`, *salary:* `real`)
> Works(*eid:* `integer`, *did:* `integer`, *pct_time:* `integer`)
> Dept(*did:* `integer`, *budget:* `real`, *managerid:* `integer`)

1. Define a table constraint on Emp that will ensure that every employee makes at least $10,000.
2. Define a table constraint on Dept that will ensure that all managers have $age > 30$.
3. Define an assertion on Dept that will ensure that all managers have $age > 30$. Compare this assertion with the equivalent table constraint. Explain which is better.
4. Write SQL statements to delete all information about employees whose salaries exceed that of the manager of one or more departments that they work in. Be sure to ensure that all the relevant integrity constraints are satisfied after your updates.

Answer 5.7 The answers are given below:

1. Define a table constraint on Emp that will ensure that every employee makes at least $10,000

```
CREATE TABLE Emp ( eid     INTEGER,
                   ename   CHAR(10),
                   age     INTEGER ,
                   salary  REAL,
                   PRIMARY KEY (eid),
                   CHECK ( salary >= 10000 ))
```

2. Define a table constraint on Dept that will ensure that all managers have $age > 30$

```
CREATE TABLE Dept ( did        INTEGER,
                    buget      REAL,
                    managerid  INTEGER ,
                    PRIMARY KEY (did),
                    FOREIGN KEY (managerid) REFERENCES Emp,
                    CHECK (    ( SELECT E.age FROM Emp E, Dept D)
                               WHERE E.eid = D.managerid ) > 30 )
```

3. Define an assertion on Dept that will ensure that all managers have age > 30

```
CREATE TABLE Dept ( did        INTEGER,
                    budget     REAL,
                    managerid  INTEGER ,
                    PRIMARY KEY (did) )


CREATE ASSERTION managerAge
CHECK ((SELECT E.age
        FROM   Emp E, Dept D
        WHERE  E.eid = D.managerid ) > 30 )
```

   Since the constraint involves two relations, it is better to define it as an assertion, independent of any one relation, rather than as a check condition on the Dept relation. The limitation of the latter approach is that the condition is checked only when the Dept relation is being updated. However, since age is an attribute of the Emp relation, it is possible to update the age of a manager which violates the constraint. So the former approach is better since it checks for potential violation of the assertion whenever one of the relations is updated.

4. To write such statements, it is necessary to consider the constraints defined over the tables. We will assume the following:

```
CREATE TABLE Emp (   eid        INTEGER,
                     ename      CHAR(10),
                     age        INTEGER,
                     salary     REAL,
                     PRIMARY KEY (eid) )

CREATE TABLE Works ( eid        INTEGER,
                     did        INTEGER,
                     pcttime    INTEGER,
                     PRIMARY KEY (eid, did),
                     FOREIGN KEY (did) REFERENCES Dept,
                     FOREIGN KEY (eid) REFERENCES Emp,
                     ON DELETE CASCADE)

CREATE TABLE Dept (  did        INTEGER,
                     buget      REAL,
                     managerid  INTEGER ,
                     PRIMARY KEY (did),
                     FOREIGN KEY (managerid) REFERENCES Emp,
                     ON DELETE SET NULL)
```

Now, we can define statements to delete employees who make more than one of their managers:

```
DELETE
FROM     Emp E
WHERE    E.eid IN ( SELECT W.eid
                    FROM   Work W, Emp E2, Dept D
                    WHERE  W.did = D.did
                    AND    D.managerid = E2.eid
                    AND    E.salary > E2.salary )
```

Exercise 5.8 Consider the following relations:

Student(*snum:* integer, *sname:* string, *major:* string,
level: string, *age:* integer)
Class(*name:* string, *meets_at:* time, *room:* string, *fid:* integer)
Enrolled(*snum:* integer, *cname:* string)
Faculty(*fid*: integer, *fname:* string, *deptid:* integer)

The meaning of these relations is straightforward; for example, Enrolled has one record per student-class pair such that the student is enrolled in the class.

1. Write the SQL statements required to create the above relations, including appropriate versions of all primary and foreign key integrity constraints.

2. Express each of the following integrity constraints in SQL unless it is implied by the primary and foreign key constraint; if so, explain how it is implied. If the constraint cannot be expressed in SQL, say so. For each constraint, state what operations (inserts, deletes, and updates on specific relations) must be monitored to enforce the constraint.

   (a) Every class has a minimum enrollment of 5 students and a maximum enrollment of 30 students.

   (b) At least one class meets in each room.

   (c) Every faculty member must teach at least two courses.

   (d) Only faculty in the department with *deptid=33* teach more than three courses.

   (e) Every student must be enrolled in the course called Math101.

   (f) The room in which the earliest scheduled class (i.e., the class with the smallest *meets_at* value) meets should not be the same as the room in which the latest scheduled class meets.

   (g) Two classes cannot meet in the same room at the same time.

(h) The department with the most faculty members must have fewer than twice the number of faculty members in the department with the fewest faculty members.

(i) No department can have more than 10 faculty members.

(j) A student cannot add more than two courses at a time (i.e., in a single update).

(k) The number of CS majors must be more than the number of Math majors.

(l) The number of distinct courses in which CS majors are enrolled is greater than the number of distinct courses in which Math majors are enrolled.

(m) The total enrollment in courses taught by faculty in the department with *deptid=33* is greater than the number of Math majors.

(n) There must be at least one CS major if there are any students whatsoever.

(o) Faculty members from different departments cannot teach in the same room.

Answer 5.8 Answer omitted.

Exercise 5.9 Discuss the strengths and weaknesses of the trigger mechanism. Contrast triggers with other integrity constraints supported by SQL.

Answer 5.9 A trigger is a procedure that is automatically invoked in response to a specified change to the database. The advantages of the trigger mechanism include the ability to perform an action based on the result of a query condition. The set of actions that can be taken is a superset of the actions that integrity constraints can take (i.e. report an error). Actions can include invoking new update, delete, or insert queries, perform data definition statements to create new tables or views, or alter security policies. Triggers can also be executed before or after a change is made to the database (that is, use old or new data).

There are also disadvantages to triggers. These include the added complexity when trying to match database modifications to trigger events. Also, integrity constraints are incorporated into database performance optimization; it is more difficult for a database to perform automatic optimization with triggers. If database consistency is the primary goal, then integrity constraints offer the same power as triggers. Integrity constraints are often easier to understand than triggers.

Exercise 5.10 Consider the following relational schema. An employee can work in more than one department; the *pct_time* field of the Works relation shows the percentage of time that a given employee works in a given department.

Emp(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`)
Works(<u>*eid:* `integer`, *did:* `integer`</u>, *pct_time:* `integer`)
Dept(<u>*did:* `integer`</u>, *budget:* `real`, *managerid:* `integer`)

Write SQL-92 integrity constraints (domain, key, foreign key, or `CHECK` constraints; or assertions) or SQL:1999 triggers to ensure each of the following requirements, considered independently.

1. Employees must make a minimum salary of $1,000.
2. Every manager must be also be an employee.
3. The total percentage of all appointments for an employee must be under 100%.
4. A manager must always have a higher salary than any employee that he or she manages.
5. Whenever an employee is given a raise, the manager's salary must be increased to be at least as much.
6. Whenever an employee is given a raise, the manager's salary must be increased to be at least as much. Further, whenever an employee is given a raise, the department's budget must be increased to be greater than the sum of salaries of all employees in the department.

Answer 5.10 Answer omitted.

6

QUERY-BY-EXAMPLE (QBE)

Exercise 6.1 Consider the following relational schema. An employee can work in more than one department.

Emp(*eid:* `integer`, *ename:* `string`, *salary:* `real`)
Works(*eid:* `integer`, *did:* `integer`)
Dept(*did:* `integer`, *dname:* `string`, *managerid:* `integer`, *floornum:* `integer`)

Write the following queries in QBE. Be sure to underline your variables to distinguish them from your constants.

1. Print the names of all employees who work on the 10th floor and make less than \$50,000.
2. Print the names of all managers who manage three or more departments on the same floor.
3. Print the names of all managers who manage 10 or more departments on the same floor.
4. Give every employee who works in the toy department a 10 percent raise.
5. Print the names of the departments that employee Santa works in.
6. Print the names and salaries of employees who work in both the toy department and the candy department.
7. Print the names of employees who earn a salary that is either less than \$10,000 or more than \$100,000.
8. Print all of the attributes for employees who work in some department that employee Santa also works in.
9. Fire Santa.

10. Print the names of employees who make more than $20,000 and work in either the video department or the toy department.

11. Print the names of all employees who work on the floor(s) where Jane Dodecahedron works.

12. Print the name of each employee who earns more than the manager of the department that he or she works in.

13. Print the name of each department that has a manager whose last name is Psmith and who is neither the highest-paid nor the lowest-paid employee in the department.

Answer 6.1 1. Names of all employees who work on the 10th floor and make less than 50,000:

Emp	*eid*	*ename*	*salary*
	_E	`P.`	$< 50,000$

Works	*eid*	*did*
	_E	_D

Dept	*did*	*dname*	*managerid*	*floor*
	_D			10

2. Names of all managers who manage three or more departments on the same floor:

Emp	*eid*	*ename*	*salary*
	_M	`P.`_N	

Dept	*did*	*dname*	*managerid*	*floor*
	_D1		_M	_F
	_D2		_M	_F
	_D3		_M	_F

Conditions
_D1 $\neq$ _D2 `AND` _D1 $\neq$ _D3 `AND` _D2 $\neq$ _D3

3. Print the names of all managers who manage ten or more departments on the same floor:

Emp	*eid*	*ename*	*salary*
	_M	`P.G.`_N	

Dept	*did*	*dname*	*managerid*	*floor*	
	_D		G._M	G.	COUNT._D>= 10

4. Give every employee who works in the Toy department a 10% raise:

Emp	*eid*	*ename*	*salary*
	_E		U._S*1.1

Works	*eid*	*did*
	_E	_D

Dept	*did*	*dname*	*managerid*	*floor*
	_D	Toy		

5. Print the names of the departments that employee Santa works in:

Emp	*eid*	*ename*	*salary*
	_E	Santa	

Works	*eid*	*did*
	_E	_D

Dept	*did*	*dname*	*managerid*	*floor*
	_D	P._N		

6. Print the names and salaries of employees who work in both the Toy department and the Candy department:

Emp	*eid*	*ename*	*salary*
UNQ.	_E	P._EN	P._ES

Works	*eid*	*did*
	_E	_D

Dept	*did*	*dname*	*managerid*	*floor*
	_D	_DN1		
	_D	_DN2		

Conditions
_DN1=Toy AND _DN2=Candy

7. Print the names of employees who earn a salary that is either less than 10,000 or more than 100,000:

Emp	*eid*	*ename*	*salary*	
		P._N	_S	_S< 10,000 OR _S> 100,000

8. Print all of the attributes for employees who work in some department that employee Santa also works in:

Emp	*eid*	*ename*	*salary*	
	_E	Santa		
P.	_E1			_E¬_E1

Works	*eid*	*did*
	_E	_D
	_E1	_D

9. Fire Santa:

Emp	*eid*	*ename*	*salary*
D.	_E	Santa	

Works	*eid*	*did*
D.	_E	

10. Print the names of employees who make more than 20,000 and work in either the Video department or the Toy department:

Emp	*eid*	*ename*	*salary*
	_E	P._EN	>20000

Works	*eid*	*did*
	_E	_D

Dept	*did*	*dname*	*managerid*	*floor*
	_D	_DN		

Conditions
_DN=Video OR _DN=Toy

11. Print the names of all employees who work on the floor(s) where Jane Dodecahedron works:

Emp	*eid*	*ename*	*salary*
	_E1	Jane Dodecahedron	
	_E2	P._N	

Works	*eid*	*did*
	_E1	_D1
	_E2	_D2

Dept	*did*	*dname*	*managerid*	*floor*
	_D1			_F
	_D2			_F

12. Print the name of each employee who earns more than the manager of the department that he or she works in:

Emp	*eid*	*ename*	*salary*
	_M		_S
	_E	P._N	>_S

Works	*eid*	*did*
	_E	_D

Dept	*did*	*dname*	*managerid*	*floor*
	_D		_M	

13. Print the name of each department that has a manager whose last name is Psmith and who is neither the highest-paid nor the lowest-paid employee in the department:

Works	*eid*	*did*
	_E	G._D

Emp	*eid*	*ename*	*salary*	
	_E		_S	
	_M	_NLIKE '%.Psmith'	_S2	_S2>_A AND _S2<_B

Temp	*did*	*Min*	*Max*
I.	_D	MIN._S	MAX._S
	_D2	_A	_B

Dept	*did*	*dname*	*managerid*	*floor*
	_D2	P._DN	_M	

Exercise 6.2 Write the following queries in QBE, based on this schema:

Suppliers(<u>*sid:* `integer`</u>, *sname:* `string`, *city:* `string`)
Parts(<u>*pid:* `integer`</u>, *pname:* `string`, *color:* `string`)
Orders(<u>*sid:* `integer`, *pid:* `integer`</u>, *quantity:* `integer`)

1. For each supplier from whom all of the following things have been ordered in quantities of at least 150, print the name and city of the supplier: a blue gear, a red crankshaft, and a yellow bumper.
2. Print the names of the purple parts that have been ordered from suppliers located in Madison, Milwaukee, or Waukesha.
3. Print the names and cities of suppliers who have an order for more than 150 units of a yellow or purple part.
4. Print the *pid*s of parts that have been ordered from a supplier named American but have also been ordered from some supplier with a different name in a quantity that is greater than the American order by at least 100 units.
5. Print the names of the suppliers located in Madison. Could there be any duplicates in the answer?
6. Print all available information about suppliers that supply green parts.
7. For each order of a red part, print the quantity and the name of the part.
8. Print the names of the parts that come in both blue and green. (Assume that no two distinct parts can have the same name and color.)
9. Print (in ascending order alphabetically) the names of parts supplied both by a Madison supplier and by a Berkeley supplier.
10. Print the names of parts supplied by a Madison supplier, but not supplied by any Berkeley supplier. Could there be any duplicates in the answer?
11. Print the total number of orders.
12. Print the largest quantity per order for each *sid* such that the minimum quantity per order for that supplier is greater than 100.
13. Print the average quantity per order of red parts.
14. Can you write this query in QBE? If so, how?
    Print the sid*s of suppliers from whom every part has been ordered.*

Answer 6.2 Answer omitted.

Exercise 6.3 Answer the following questions:

1. Describe the various uses for unnamed columns in QBE.
2. Describe the various uses for a conditions box in QBE.
3. What is unusual about the treatment of duplicates in QBE?
4. Is QBE based upon relational algebra, tuple relational calculus, or domain relational calculus? Explain briefly.
5. Is QBE relationally complete? Explain briefly.
6. What restrictions does QBE place on update commands?

Answer 6.3 1. If we want to display some information in addition to fields retrieved from a relation, we can do this by creating *unnamed fields* for display. For example:

Sailors	*sid*	*sname*	*rating*	*age*	
		`P.`	_R	_A	`P.`_R / _A

If we want to display fields from more than one table, we can use unnamed columns. For example:

Sailors	*sid*	*sname*	*rating*	*age*	
	_Id	`P.`			`P.`_D

Reserves	*sid*	*bid*	*date*
	_Id		_D

2. Conditions boxes are used to do the following:

 - *Express a condition involving two or more columns*, e.g., '_R / _A >0.2'.
 - *Express a condition involving an aggregate operation.* This is similiar to the `HAVING` clause in SQL. For example:

Suppliers	*sid*	*sname*	*city*
	`P.`_S		`G.P.`_C

Conditions
`COUNT.`_S>5

 - *Express conditions involving the* `AND` *and* `OR` *operators.* For Example:

Suppliers	*sid*	*sname*	*city*
	`P.`_S		_C

Conditions
_C=Madison `OR` _C=Milwaukee `OR` _C=Waukesha

3. The default treatment of duplicates in QBE is unusual. If the query contains a single row with `P.`, the default is that duplicates are not eliminated. If the query contains more than one such row, duplicates are eliminated by default. In either case, you can explicitly specify whether duplicates are to be eliminated (or not) by putting `ALL.` (resp. `UNQ.`) under the relation name. For Example:

Sailors	*sid*	*sname*	*rating*	*age*
UNQ.		P._N		P._A

On the following query, duplicates are eliminated by default, and the name of each sailor in this age range is printed once.

Sailors	*sid*	*sname*	*rating*	*age*
		P._S		< 30
		P._S		> 20

Putting ALL. in the first column (of one of the rows) results in printing the names of qualifying sailors as often as there are sailors with this name in the given age range.

4. Yes, QBE is based upon Doman Relational Calculus. A user writes queries by creating *example tables*. QBE uses *domain variables*, as in the domain relational calculus (DRC), to create example tables. The domain of a variable is determined by the column in which it appears, and variable symbols are prefixed with '_' to distinguish them from constants. Constants, including strings, appear unquoted, in contrast to SQL. The fields that should appear in the answer are specified by using the command 'P.', which stands for 'print'. The fields containing this command are analoguous to the *target-list* in the SELECT clause of an SQL query.

5. Yes. QBE cannot accomplish some Queries without the use of aggregate operator, unless it make use of the update commands to create a temporary relation or view. Therefore, taking the update commands into account, QBE is relationally complete, even without the aggregate operators.

   To understand the difficulty of expressing division in QBE, consider the following query: *Find sailors who have reserved all boats.*

Sailors	*sid*	*sname*	*rating*	*age*
	_Id			

Boats	*bid*	*bname*	*color*
	_B		

Reserves	*sid*	*bid*	*date*
¬	_Id	_B	

BadSids	*sid*
I.	_Id

Given the view BadSids, it is a simple matter to find sailors whose *sid*'s are not in this view.

The ideas in this example can be extended to show that QBE is *relationally complete*.

6. There are some restrictions on the use of the `I.`, `D.` and `U.` commands. First, we cannot mix these operators in a single example table (or combine them with `P.`). Second, we cannot specify `I.`, `D.` or `U.` in an example table that contains `G.`. Third, we cannot insert, update, or modify tuples based upon values in fields of other tuples (in the same table, or different tables). Thus, the following update is incorrect:

Sailors	*sid*	*sname*	*rating*	*age*
		john		U._A1
		joe		_A

7

STORING DATA: DISKS AND FILES

Exercise 7.1 What is the most important difference between a disk and a tape?

Answer 7.1 *Tapes* are sequential devices that do not support direct access to a desired page. We must essentially step through all pages in order. *Disks* support direct access to a desired page.

Exercise 7.2 Explain the terms *seek time, rotational delay*, and *transfer time.*

Answer 7.2 Answer omitted.

Exercise 7.3 Both disks and main memory support direct access to any desired location (page). On average, main memory accesses are faster, of course. What is the other important difference (from the perspective of the time required to access a desired page)?

Answer 7.3 The time to access a disk page is not constant. It depends on the location of the data. Accessing to some data might be much faster than to others. It is different for memory. Access to memory is uniform for most computer systems.

Exercise 7.4 If you have a large file that is frequently scanned sequentially, explain how you would store the pages in the file on a disk.

Answer 7.4 Answer omitted.

Exercise 7.5 Consider a disk with a sector size of 512 bytes, 2,000 tracks per surface, 50 sectors per track, 5 double-sided platters, average seek time of 10 msec.

1. What is the capacity of a track in bytes? What is the capacity of each surface? What is the capacity of the disk?

2. How many cylinders does the disk have?
3. Give examples of valid block sizes. Is 256 bytes a valid block size? 2,048? 51,200?
4. If the disk platters rotate at 5,400 rpm (revolutions per minute), what is the maximum rotational delay?
5. Assuming that one track of data can be transferred per revolution, what is the transfer rate?

Answer 7.5 1.

$$bytes/track = bytes/sector \times sectors/track = 512 \times 50 = 25K$$

$$bytes/surface = bytes/track \times tracks/surface = 25K \times 2000 = 50,000K$$

$$bytes/disk = bytes/surface \times surfaces/disk = 50,000K \times 10 = 500,000K$$

2. The number of cylinders is the same as the number of tracks on each platter, which is 2000.
3. The block size should be a multiple of the sector size. We can see that 256 is not a valid block size while 2048 and 51200 are.
4. If the disk platters rotate at 5400rpm, the time required for a rotation, which is the maximum rotational delay, is

$$\frac{1}{5400} \times 60 = 0.011seconds$$

   . The average rotational delay is half of the rotation time, 0.006 seconds.
5. The capacity of a track is 25K bytes. Since one track of data can be transferred per revolution, the data transfer rate is

$$\frac{25K}{0.011} = 2,250Kbytespersec$$

Exercise 7.6 Consider again the disk specifications from Exercise 7.5 and suppose that a block size of 1,024 bytes is chosen. Suppose that a file containing 100,000 records of 100 bytes each is to be stored on such a disk and that no record is allowed to span two blocks.

1. How many records fit onto a block?
2. How many blocks are required to store the entire file? If the file is arranged sequentially on disk, how many surfaces are needed?
3. How many records of 100 bytes each can be stored using this disk?

4. If pages are stored sequentially on disk, with page 1 on block 1 of track 1, what is the page stored on block 1 of track 1 on the next disk surface? How would your answer change if the disk were capable of reading/writing from all heads in parallel?

5. What is the time required to read a file containing 100,000 records of 100 bytes each sequentially? Again, how would your answer change if the disk were capable of reading/writing from all heads in parallel (and the data was arranged optimally)?

6. What is the time required to read a file containing 100,000 records of 100 bytes each in some random order? Note that in order to read a record, the block containing the record has to be fetched from disk. Assume that each block request incurs the average seek time and rotational delay.

Answer 7.6 Answer omitted.

Exercise 7.7 Explain what the buffer manager must do to process a read request for a page. What happens if the requested page is in the pool but not pinned?

Answer 7.7 When a page is requested the buffer manager does the following:

1. The buffer pool is checked to see if it contains the requested page. If the page is not in the pool, it is brought in as follows:
   (a) A frame is chosen for replacement, using the replacement policy.
   (b) If the frame chosen for replacement is dirty, it is *flushed* (the page it contains is written out to disk).
   (c) The requested page is read into the frame chosen for replacement.
2. The requested page is *pinned* (the *pin_count* of its frame is incremented) and its address is returned to the requestor.

Note that if the page is not pinned,it could be removed from buffer pool even if it is actually needed in main memory.

Exercise 7.8 When does a buffer manager write a page to disk?

Answer 7.8 Answer omitted.

Exercise 7.9 What does it mean to say that a page is *pinned* in the buffer pool? Who is responsible for pinning pages? Who is responsible for unpinning pages?

Answer 7.9

1. *Pinning* a page means the *pin_count* of its frame is incremented. Pinning a page guarantees higher-level DBMS software that the page will not be removed from the buffer pool by the buffer manager. That is, another file page will not be read into the frame containing this page until it is unpinned by this requestor.
2. It is the buffer manager's responsibility to pin a page.
3. It is the responsibility of the requestor of that page to tell the buffer manager to unpin a page.

Exercise 7.10 When a page in the buffer pool is modified, how does the DBMS ensure that this change is propagated to disk? (Explain the role of the buffer manager as well as the modifier of the page.)

Answer 7.10 Answer omitted.

Exercise 7.11 What happens if there is a page request when all pages in the buffer pool are dirty?

Answer 7.11 If there are some unpinned pages, the buffer manager chooses one by using a *replacement policy*, flushes this page, and then replaces it with the requested page.

If there are no unpinned pages, the buffer manager has to wait until an unpinned page is available (or signal an error condition to the page requestor).

Exercise 7.12 What is *sequential flooding* of the buffer pool?

Answer 7.12 Answer omitted.

Exercise 7.13 Name an important capability of a DBMS buffer manager that is not supported by a typical operating system's buffer manager.

Answer 7.13

1. Pinning a page to prevent it from being replaced.
2. Ability to explicitly force a single page to disk.

Exercise 7.14 Explain the term *prefetching*. Why is it important?

Answer 7.14 Answer omitted.

Exercise 7.15 Modern disks often have their own main memory caches, typically about one MB, and use this to do prefetching of pages. The rationale for this technique is the empirical observation that if a disk page is requested by some (not necessarily database!) application, 80 percent of the time the next page is requested as well. So the disk gambles by reading ahead.

1. Give a nontechnical reason that a DBMS may not want to rely on prefetching controlled by the disk.
2. Explain the impact on the disk's cache of several queries running concurrently, each scanning a different file.
3. Can the above problem be addressed by the DBMS buffer manager doing its own prefetching? Explain.
4. Modern disks support *segmented caches*, with about four to six segments, each of which is used to cache pages from a different file. Does this technique help, with respect to the above problem? Given this technique, does it matter whether the DBMS buffer manager also does prefetching?

Answer 7.15

1. The pre-fetching done at the disk level varies widely across different drives and manufacturers, and pre-fetching is sufficiently important to a DBMS that one would like to be independent of specific hardware support.
2. If there are many queries running concurrently, the request of a page from different queries can be interleaved. In the worst case, it cause the cache miss on every page request, even with disk pre-fetching.
3. If we have pre-fetching offered by DBMS buffer manager, the buffer manager can predict the reference pattern more accurately. In particular, a certain number of buffer frames can be allocated *per* active scan for pre-fetching purposes, and interleaved requests would not compete for the same frames.

Exercise 7.16 Describe two possible record formats. What are the trade-offs between them?

Answer 7.16 Answer omitted.

Exercise 7.17 Describe two possible page formats. What are the trade-offs between them?

Answer 7.17 Two possible page formats are: *consecutive slots* and *slot directory*

The consecutive slots organization is mostly used for fixed length record formats. It handles the deletion by using bitmaps or linked lists.

The slot directory organization maintains a directory of slots for each page, with a ¡*record offset*, *record length*¿ pair per slot.

The slot directory is an indirect way to get the offset of a entry. Because of this indirection, deletion is easy. It is accomplished by setting the length field to 0. And records can easily be moved around on the page without changing their external identifier.

Exercise 7.18 Consider the page format for variable-length records that uses a slot directory.

1. One approach to managing the slot directory is to use a maximum size (i.e., a maximum number of slots) and to allocate the directory array when the page is created. Discuss the pros and cons of this approach with respect to the approach discussed in the text.
2. Suggest a modification to this page format that would allow us to sort records (according to the value in some field) without moving records and without changing the record ids.

Answer 7.18 Answer omitted.

Exercise 7.19 Consider the two internal organizations for heap files (using lists of pages and a directory of pages) discussed in the text.

1. Describe them briefly and explain the trade-offs. Which organization would you choose if records are variable in length?
2. Can you suggest a single page format to implement both internal file organizations?

Answer 7.19 1. The list of pages in shown in Fig 3.7. The directory of pages is shown in Fig 3.8.

2. The linked-list approach is a little simpler, but finding a page with sufficient free space for a new record (especially with variable length records) is harder. We have to essentially scan the list of pages until we find one with enough space, whereas the directory organization allows us to find such a page by simply scanning the directory, which is much smaller than the entire file. The directory organization is therefore better, especially with variable length records.
3. A page format with *previous* and *next* page pointers would help in both cases. Obviously, such a page format allows us to build the linked list organization; it is also useful for implementing the directory in the directory organization.

Exercise 7.20 Consider a list-based organization of the pages in a heap file in which two lists are maintained: a list of *all* pages in the file and a list of all pages with free space. In contrast, the list-based organization discussed in the text maintains a list of full pages and a list of pages with free space.

1. What are the trade-offs, if any? Is one of them clearly superior?
2. For each of these organizations, describe a page format that can be used to implement it.

Answer 7.20 Answer omitted.

Exercise 7.21 Modern disk drives store more sectors on the outer tracks than the inner tracks. Since the rotation speed is constant, the sequential data transfer rate is also higher on the outer tracks. The seek time and rotational delay are unchanged. Considering this information, explain good strategies for placing files with the following kinds of access patterns:

1. Frequent, random accesses to a small file (e.g., catalog relations).
2. Sequential scans of a large file (e.g., selection from a relation with no index).
3. Random accesses to a large file via an index (e.g., selection from a relation via the index).
4. Sequential scans of a small file.

Answer 7.21

1. Place the file in the middle tracks. Sequential speed is not an issue due to the small size of the file, and the seek time is minimized by placing files in the center.
2. Place the file in the outer tracks. Sequential speed is most important and outer tracks maximize it.
3. Place the file and index on the inner tracks. The DBMS will alternately access pages of the index and of the file, and so the two should reside in close proximity to reduce seek times. By placing the file and the index on the inner tracks we also save valuable space on the faster (outer) tracks for other files that are accessed sequentially.
4. Place small files in the inner half of the disk. A scan of a small file is effectively random I/O because the cost is dominated by the cost of the initial seek to the beginning of the file.

8

FILE ORGANIZATIONS AND INDEXES

Exercise 8.1 What are the main conclusions that you can draw from the discussion of the three file organizations?

Answer 8.1 The main conclusion about the three file organizations is that all three file organizations have their own advantages and disadvantages. No one file organization is uniformly superior in all situations. The choice of appropriate structures for a given data set can have a significant impact upon performance. An unordered file is best if only full file scans are desired. A hashed file is best if the most common operation is an equality selection. A sorted file is best (of the three alternatiaves considered in this chapter) if range selections are desired.

Exercise 8.2 Consider a delete specified using an equality condition. What is the cost if no record qualifies? What is the cost if the condition is not on a key?

Answer 8.2 Answer omitted.

Exercise 8.3 Which of the three basic file organizations would you choose for a file where the most frequent operations are as follows?

1. Search for records based on a range of field values.
2. Perform inserts and scans where the order of records does not matter.
3. Search for a record based on a particular field value.

Answer 8.3

1. Using these fields as the search key, we would choose a sorted file organization.
2. Heap file would be the best fit in this situation.
3. Using this particular field as the searach key, choosing a hashed file would be the best.

Exercise 8.4 Explain the difference between each of the following:

1. Primary versus secondary indexes.
2. Dense versus sparse indexes.
3. Clustered versus unclustered indexes.

If you were about to create an index on a relation, what considerations would guide your choice with respect to each pair of properties listed above?

Answer 8.4 Answer omitted.

Exercise 8.5 Consider a relation stored as a randomly ordered file for which the only index is an unclustered index on a field called *sal*. If you want to retrieve all records with $sal > 20$, is using the index always the best alternative? Explain.

Answer 8.5 No. In this case, the index is unclustered, each qualifying data entries could contain an rid that points to a distinct data page, leading to as many data page I/Os as the number of data entries that match the range query. At this time,using index is worse than file scan.

Exercise 8.6 If an index contains data records as 'data entries', is it clustered or unclustered? Dense or sparse?

Answer 8.6 Answer omitted.

Exercise 8.7 Consider Alternatives (1), (2) and (3) for 'data entries' in an index, as discussed in Section 8.3.1. Are they all suitable for secondary indexes? Explain.

Answer 8.7 Yes.All the three alternatives allow duplicate data entries.

Exercise 8.8 Consider the instance of the Students relation shown in Figure 8.1, sorted by *age*: For the purposes of this question, assume that these tuples are stored in a sorted file in the order shown; the first tuple is in page 1, slot 1; the second tuple is in page 1, slot 2; and so on. Each page can store up to three data records. You can use ⟨*page-id, slot*⟩ to identify a tuple.

List the data entries in each of the following indexes. If the order of entries is significant, say so and explain why. If such an index cannot be constructed, say so and explain why.

1. A dense index on *age* using Alternative (1).

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@ee	19	3.2
53650	Smith	smith@math	19	3.8

Figure 8.1 An Instance of the Students Relation, Sorted by *age*

2. A dense index on *age* using Alternative (2).
3. A dense index on *age* using Alternative (3).
4. A sparse index on *age* using Alternative (1).
5. A sparse index on *age* using Alternative (2).
6. A sparse index on *age* using Alternative (3).
7. A dense index on *gpa* using Alternative (1).
8. A dense index on *gpa* using Alternative (2).
9. A dense index on *gpa* using Alternative (3).
10. A sparse index on *gpa* using Alternative (1).
11. A sparse index on *gpa* using Alternative (2).
12. A sparse index on *gpa* using Alternative (3).

Answer 8.8 Answer omitted.

9

TREE-STRUCTURED INDEXING

Exercise 9.1 Consider the B+ tree index of order $d = 2$ shown in Figure 9.1.

1. Show the tree that would result from inserting a data entry with key 9 into this tree.
2. Show the B+ tree that would result from inserting a data entry with key 3 into the original tree. How many page reads and page writes will the insertion require?
3. Show the B+ tree that would result from deleting the data entry with key 8 from the original tree, assuming that the left sibling is checked for possible redistribution.
4. Show the B+ tree that would result from deleting the data entry with key 8 from the original tree, assuming that the right sibling is checked for possible redistribution.
5. Show the B+ tree that would result from starting with the original tree, inserting a data entry with key 46 and then deleting the data entry with key 52.
6. Show the B+ tree that would result from deleting the data entry with key 91 from the original tree.

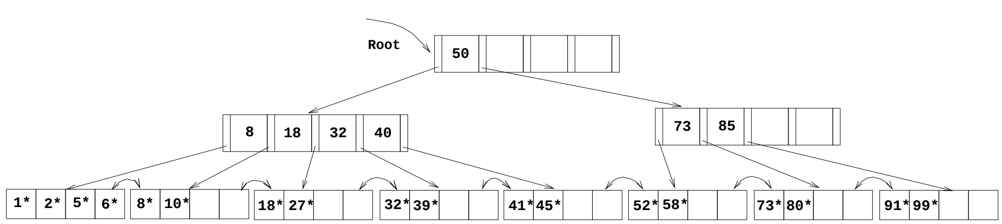

Figure 9.1 Tree for Exercise 9.1

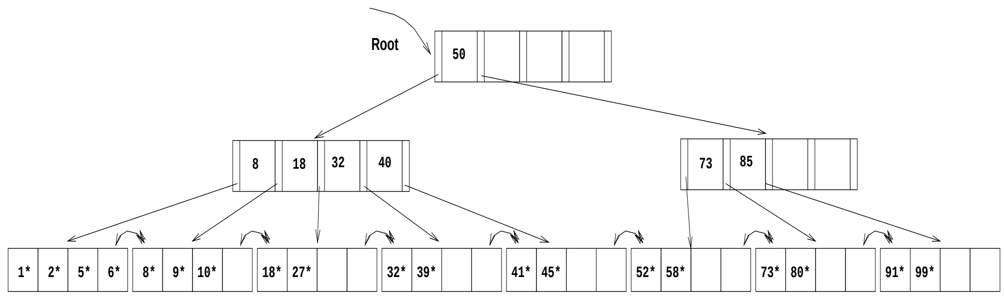

Figure 9.2

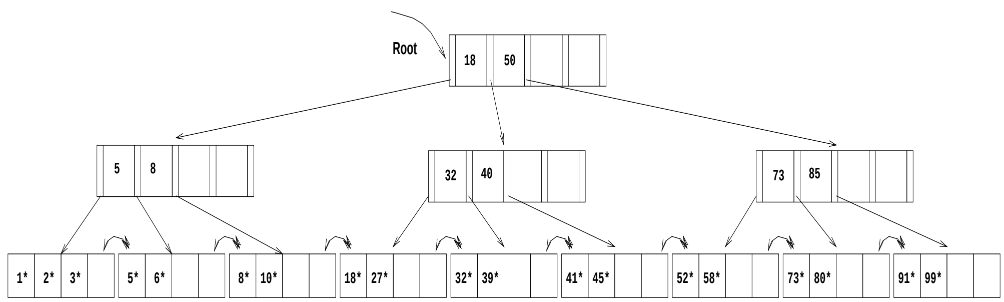

Figure 9.3

7. Show the B+ tree that would result from starting with the original tree, inserting a data entry with key 59, and then deleting the data entry with key 91.

8. Show the B+ tree that would result from successively deleting the data entries with keys 32, 39, 41, 45, and 73 from the original tree.

Answer 9.1 1. The data entry with key 9 is inserted on the second leaf page. The resulting tree is shown in figure 9.2.

2. The data entry with key 3 goes on the first leaf page F. Since F can accommodate at most four data entries ($d = 2$), F splits. The lowest data entry of the new leaf is given up to the ancestor which also splits. The result can be seen in figure 9.3. The insertion will require 6 page writes, 4 page reads and allocation of 2 new pages.

3. The data entry with key 8 is deleted, resulting in a leaf page N with less than two data entries. The left sibling L is checked for redistribution. Since L has more than two data entries, the remaining keys are redistributed between L and N, resulting in the tree in figure 9.4.

4. As is part 3, the data entry with key 8 is deleted from the leaf page N. N's right sibling R is checked for redistribution, but R has the minimum number of keys.

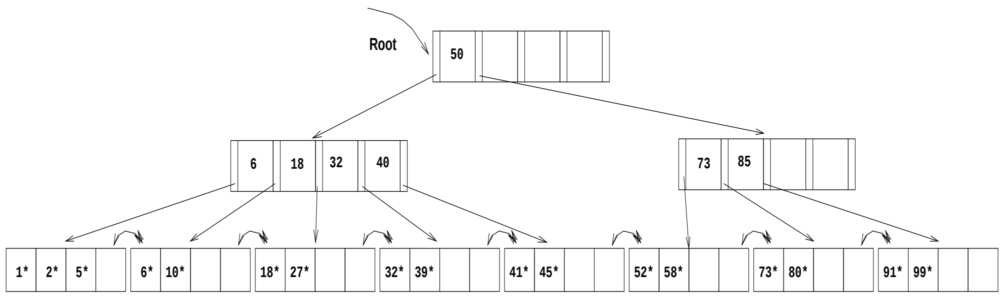

Figure 9.4

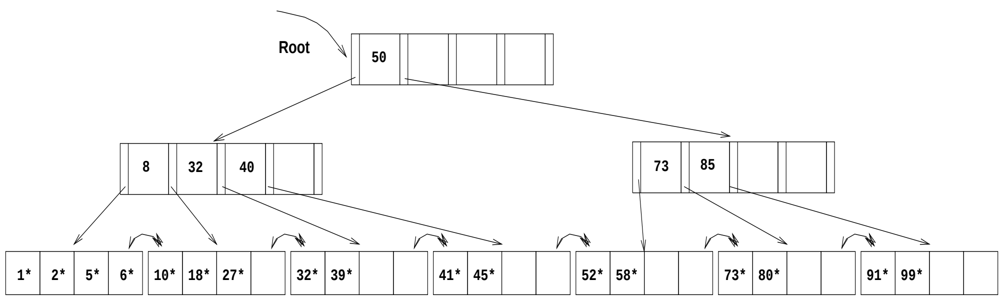

Figure 9.5

Therefore the two siblings merge. The key in the ancestor which distinguished between the newly merged leaves is deleted. The resulting tree is shown in figure 9.5.

5. The data entry with key 46 can be inserted without any structural changes in the tree. But the removal of the data entry with key 52 causes its leaf page L to merge with a sibling (we chose the right sibling). This results in the removal of a key in the ancestor A of L and thereby lowering the number of keys on A below the minimum number of keys. Since the left sibling B of A has more than the minimum number of keys, redistributon between A and B takes place. The final tree is depicted in figure 9.6.

6. Deleting the data entry with key 91 causes a scenario similar to part 5. The result can be seen in figure 9.7.

7. The data entry with key 59 can be inserted without any structural changes in the tree. No sibling of the leaf page with the data entry with key 91 is effected by the insert. Therefore deleting the data entry with key 91 changes the tree in a way very similar to part 6. The result is depicted in figure 9.8.

8. The successive deletion of the data entries with keys 32, 39, 41, 45 and 73 results in the tree in figure 9.9.

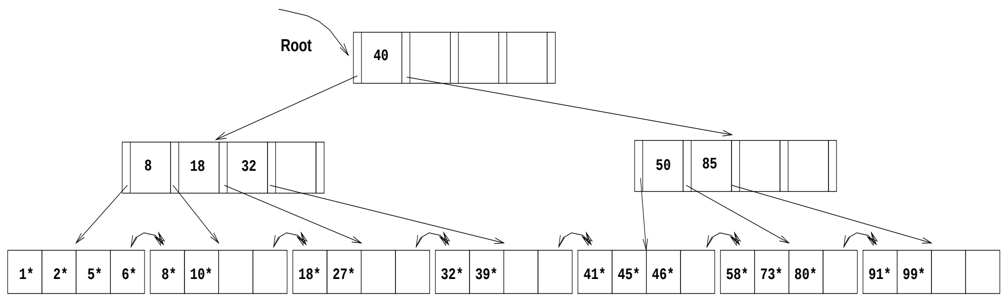

Figure 9.6

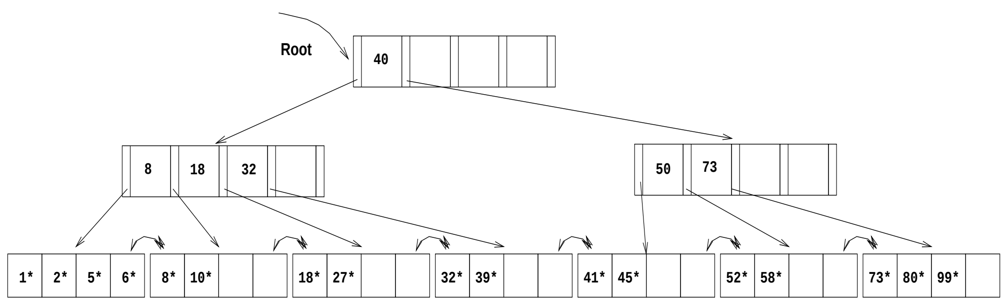

Figure 9.7

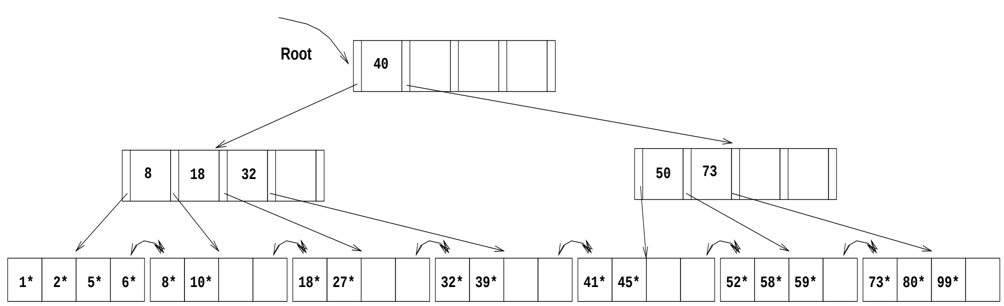

Figure 9.8

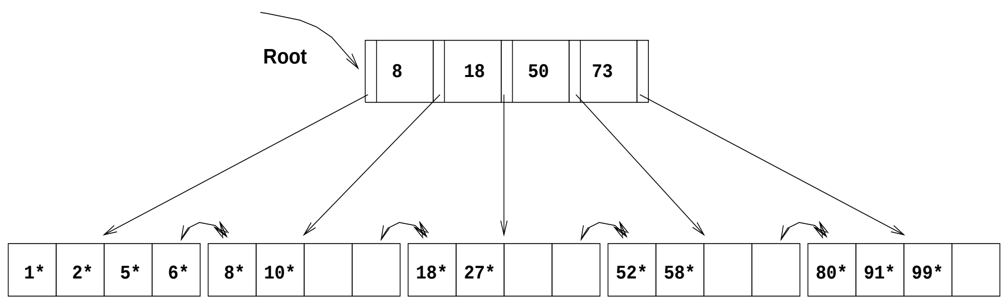

Figure 9.9

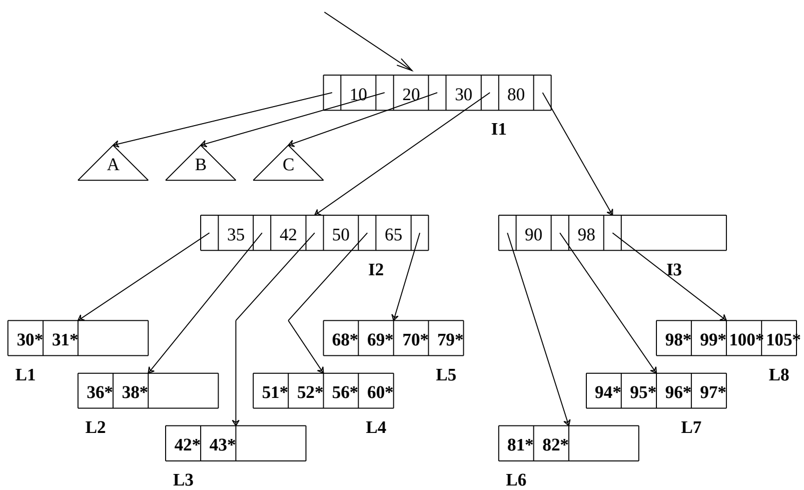

Figure 9.10 Tree for Exercise 9.2

Exercise 9.2 Consider the B+ tree index shown in Figure 9.10, which uses Alternative (1) for data entries. Each intermediate node can hold up to five pointers and four key values. Each leaf can hold up to four records, and leaf nodes are doubly linked as usual, although these links are not shown in the figure.

Answer the following questions.

1. Name all the tree nodes that must be fetched to answer the following query: "Get all records with search key greater than 38."
2. Insert a record with search key 109 into the tree.
3. Delete the record with search key 81 from the (original) tree.
4. Name a search key value such that inserting it into the (original) tree would cause an increase in the height of the tree.

5. Note that subtrees A, B, and C are not fully specified. Nonetheless, what can you infer about the contents and the shape of these trees?

6. How would your answers to the above questions change if this were an ISAM index?

7. Suppose that this is an ISAM index. What is the minimum number of insertions needed to create a chain of three overflow pages?

Answer 9.2 Answer omitted.

Exercise 9.3 Answer the following questions.

1. What is the minimum space utilization for a B+ tree index?

2. What is the minimum space utilization for an ISAM index?

3. If your database system supported both a static and a dynamic tree index (say, ISAM and B+ trees), would you ever consider using the *static* index in preference to the *dynamic* index?

Answer 9.3 The answer to each question is given below.

1. By the definition of a B+ tree, each index page, except for the root, has at least d and at most $2d$ key entries. Therefore—with the exception of the root—the minimum space utilization guaranteed by a B+ tree index is 50 percent.

2. The minimum space utilization by an ISAM index depends on the design of the index and the data distribution over the lifetime of ISAM index. Since an ISAM index is static, empty spaces in index pages are never filled (in contrast to a B+ tree index, which is a dynamic index). Therefore the space utilization of ISAM index pages is usually close to 100 percent by design. However, there is no guarantee for data pages' utilization.

3. A static index without overflow pages is faster than a dynamic index on inserts and deletes, since index pages are only read and never written. If the set of keys that will be inserted into the tree is known in advance, then it is possible to build a static index which reserves enough space for all possible future inserts. Also if the system goes periodically off line, static indices can be rebuilt and scaled to the current occupancy of the index; infrequent or scheduled updates are flags for when to consider a static index structure.

Exercise 9.4 Suppose that a page can contain at most four data values and that all data values are integers. Using only B+ trees of order 2, give examples of each of the following:

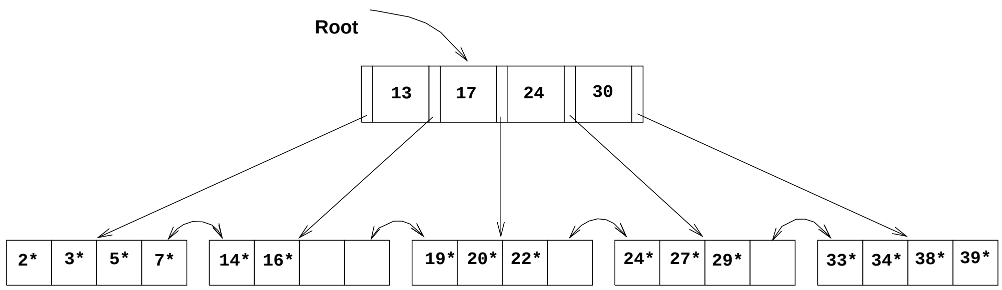

Figure 9.11 Tree for Exercise 9.5

1. A B+ tree whose height changes from 2 to 3 when the value 25 is inserted. Show your structure before and after the insertion.
2. A B+ tree in which the deletion of the value 25 leads to a redistribution. Show your structure before and after the deletion.
3. A B+ tree in which the deletion of the value 25 causes a merge of two nodes, but without altering the height of the tree.
4. An ISAM structure with four buckets, none of which has an overflow page. Further, every bucket has space for exactly one more entry. Show your structure before and after inserting two additional values, chosen so that an overflow page is created.

Answer 9.4 Answer omitted.

Exercise 9.5 Consider the B+ tree shown in Figure 9.11.

1. Identify a list of five data entries such that:
   (a) Inserting the entries in the order shown and then deleting them in the opposite order (e.g., insert a, insert b, delete b, delete a) results in the original tree.
   (b) Inserting the entries in the order shown and then deleting them in the opposite order (e.g., insert a, insert b, delete b, delete a) results in a different tree.
2. What is the minimum number of insertions of data entries with distinct keys that will cause the height of the (original) tree to change from its current value (of 1) to 3?
3. Would the minimum number of insertions that will cause the original tree to increase to height 3 change if you were allowed to insert duplicates (multiple data entries with the same key), assuming that overflow pages are not used for handling duplicates?

Answer 9.5 The answer to each question is given below.

1. The answer to each part is given below.
   (a) One example is the set of five data entries with keys 17, 18, 13, 15, and 25. Inserting 17 and 18 will cause the tree to split and gain a level. Inserting 13, 15, and 25 does change the tree structure any further, so deleting them in reverse order causes no structure change. When 18 is deleted, redistribution will be possible from an adjacent node since one node will contain only the value 17, and its right neighbor will contain 19, 20, and 22. Finally, when 17 is deleted, no redistribution will be possible so the tree will loose a level and will return to the original tree.
   (b) Inserting and deleting the set 13, 15, 18, 25, and 4 will cause a change in the tree structure. When 4 is inserted, the right most leave will split causing the tree to gain a level. When it is deleted, the tree will not shrink in size. Since inserts 13, 15, 18, and 25 did not affect the right most node, their deletion will not change the altered structure either.
2. Let us call the current tree depicted in Figure 9.11 T. T has 16 data entries. The smallest tree S of height 3 which is created exclusively through inserts has $(1*2*3*3)*2+1 = 37$ data entries in its leaf pages. S has 18 leaf pages with two data entries each and one leaf page with three data entries. T has already four leaf pages which have more than two data entries; they can be filled and made to split, but after each spilt, one of the two pages will still has three data entries remaining. Therefore the smallest tree of height 3 which can possibly be created from T only through inserts has $(1 * 2 * 3 * 3) * 2 + 4 = 40$ data entries. Therefore the minimum number of entries that will cause the height of T to change to 3 is $40 - 16$=24.
3. The argument in part 2 does not assume anything about the data entries to be inserted; it is valid if duplicates can be inserted as well. Therefore the solution does not change.

Exercise 9.6 Answer Exercise 9.5 assuming that the tree is an ISAM tree! (Some of the examples asked for may not exist—if so, explain briefly.)

Answer 9.6 Answer omitted.

Exercise 9.7 Suppose that you have a sorted file, and you want to construct a dense primary B+ tree index on this file.

1. One way to accomplish this task is to scan the file, record by record, inserting each one using the B+ tree insertion procedure. What performance and storage utilization problems are there with this approach?

2. Explain how the bulk-loading algorithm described in the text improves upon the above scheme.

Answer 9.7 1. This approach is likely to be quite expensive, since each entry requires us to start from the root and go down to the appropriate leaf page. Even though the index level pages are likely to stay in the buffer pool between successive requests, the overhead is still considerable. Also, according to the insertion algorithm, each time a node splits, the data entries are redistributed evenly to both nodes. This leads to a fixed page utilization of 50%

2. The bulk loading algorithm has good performance and space utilization compared with the repeated inserts approach. Since the B+ tree is grown from the bottom up, the bulk loading algorithm allows the administrator to pre-set the amount each index and data page should be filled. This allows good performance for future inserts, and supports some desired space utilization.

Exercise 9.8 Assume that you have just built a dense B+ tree index using Alternative (2) on a heap file containing 20,000 records. The key field for this B+ tree index is a 40-byte string, and it is a candidate key. Pointers (i.e., record ids and page ids) are (at most) 10-byte values. The size of one disk page is 1,000 bytes. The index was built in a bottom-up fashion using the bulk-loading algorithm, and the nodes at each level were filled up as much as possible.

1. How many levels does the resulting tree have?
2. For each level of the tree, how many nodes are at that level?
3. How many levels would the resulting tree have if key compression is used and it reduces the average size of each key in an entry to 10 bytes?
4. How many levels would the resulting tree have without key compression, but with all pages 70 percent full?

Answer 9.8 Answer omitted.

Exercise 9.9 The algorithms for insertion and deletion into a B+ tree are presented as recursive algorithms. In the code for *insert*, for instance, there is a call made at the parent of a node N to insert into (the subtree rooted at) node N, and when this call returns, the current node is the parent of N. Thus, we do not maintain any 'parent pointers' in nodes of B+ tree. Such pointers are not part of the B+ tree structure for a good reason, as this exercise will demonstrate. An alternative approach that uses parent pointers—again, remember that such pointers are *not* part of the standard B+ tree structure!—in each node appears to be simpler:

Search to the appropriate leaf using the search algorithm; then insert the entry and split if necessary, with splits propagated to parents if necessary (using the parent pointers to find the parents).

Consider this (unsatisfactory) alternative approach:

1. Suppose that an internal node N is split into nodes N and N2. What can you say about the parent pointers in the children of the original node N?
2. Suggest two ways of dealing with the inconsistent parent pointers in the children of node N.
3. For each of the above suggestions, identify a potential (major) disadvantage.
4. What conclusions can you draw from this exercise?

Answer 9.9 The answer to each question is given below.

1. The parent pointers in either d or $d+1$ of the children of the original node N are not valid any more: they still point to N, but they should point to $N2$.
2. One solution is to adjust all parent pointers in the children of the original node N which became children of $N2$. Another solution is to leave the pointers during the insert operation and to adjust them later when the page is actually needed and read into memory anyway.
3. The first solution requires at least $d+1$ additional page reads (and sometime later, page writes) on an insert, which would result in a remarkable slowdown. In the second solution mentioned above, a child M, which has a parent pointer to be adjusted, is updated if an operation is performed which actually reads M into memory (maybe on a down path from the root to a leaf page). But this solution modifies M and therefore requires sometime later a write of M, which might not have been necessary if there were no parent pointers.
4. In conclusion, to add parent pointers to the B+ tree data structure is not a good modification. Parent pointers cause unnecessary page updates and so lead to a decrease in performance.

Exercise 9.10 Consider the instance of the Students relation shown in Figure 9.12. Show a B+ tree of order 2 in each of these cases, assuming that duplicates are handled using overflow pages. Clearly indicate what the data entries are (i.e., do not use the '$k*$' convention).

1. A dense B+ tree index on *age* using Alternative (1) for data entries.

sid	*name*	*login*	*age*	*gpa*
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	3.8
53666	Jones	jones@cs	18	3.4
53901	Jones	jones@toy	18	3.4
53902	Jones	jones@physics	18	3.4
53903	Jones	jones@english	18	3.4
53904	Jones	jones@genetics	18	3.4
53905	Jones	jones@astro	18	3.4
53906	Jones	jones@chem	18	3.4
53902	Jones	jones@sanitation	18	3.8
53688	Smith	smith@ee	19	3.2
53650	Smith	smith@math	19	3.8
54001	Smith	smith@ee	19	3.5
54005	Smith	smith@cs	19	3.8
54009	Smith	smith@astro	19	2.2

Figure 9.12 An Instance of the Students Relation

2. A sparse B+ tree index on *age* using Alternative (1) for data entries.

3. A dense B+ tree index on *gpa* using Alternative (2) for data entries. For the purposes of this question, assume that these tuples are stored in a sorted file in the order shown in the figure: the first tuple is in page 1, slot 1; the second tuple is in page 1, slot 2; and so on. Each page can store up to three data records. You can use ⟨*page-id, slot*⟩ to identify a tuple.

Answer 9.10 Answer omitted.

Exercise 9.11 Suppose that duplicates are handled using the approach without overflow pages discussed in Section 9.7. Describe an algorithm to search for the left-most occurrence of a data entry with search key value K.

Answer 9.11 The key to understanding this problem is to observe that when a leaf splits due to inserted duplicates, then of the two resulting leaves, it may happen that the left leaf contains other search key values less than the duplicated search key value. Furthermore, it could happen that least element on the right leaf could be the duplicated value. (This scenario could arise, for example, when the majority of data entries on the original leaf were for search keys of the duplicated value.) The parent index node (assuming the tree is of at least height 2) will have an entry for the duplicated value with a pointer to the rightmost leaf.

If this leaf continues to be filled with entries having the same duplicated key value, it could split again causing another entry with the same key value to be inserted in the parent node. Thus, the same key value could appear many times in the index nodes as well. While searching for entries with a given key value, the search should proceed by using the left-most of the entries on an index page such that the key value is less than or equal to the given key value. Moreover, on reaching the leaf level, it is possible that there are entries with the given key value (call it k) on the page to the *left* of the current leaf page, unless some entry with a smaller key value is present on this leaf page. Thus, we must scan to the left using the neighbor pointers at the leaf level until we find an entry with a key value *less than* k (or come to the beginning of the leaf pages). Then, we must scan forward along the leaf level until we find an entry with a key value *greater than* k.

Exercise 9.12 Answer Exercise 9.10 assuming that duplicates are handled without using overflow pages, using the alternative approach suggested in Section 9.7.

Answer 9.12 Answer omitted.

10

HASH-BASED INDEXING

Exercise 10.1 Consider the Extendible Hashing index shown in Figure 10.1. Answer the following questions about this index:

1. What can you say about the last entry that was inserted into the index?
2. What can you say about the last entry that was inserted into the index if you know that there have been no deletions from this index so far?
3. Suppose you are told that there have been no deletions from this index so far. What can you say about the last entry whose insertion into the index caused a split?

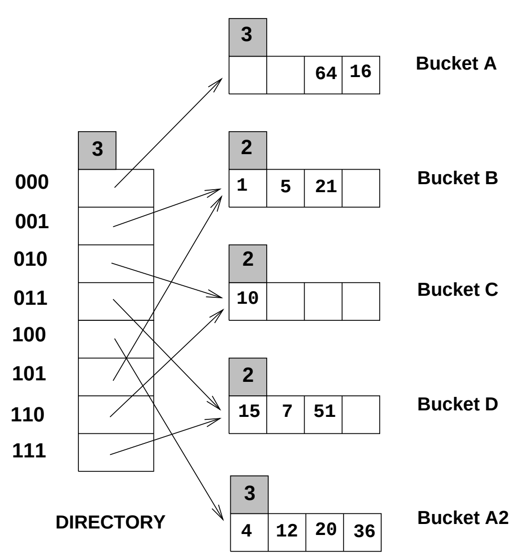

Figure 10.1 Figure for Exercise 10.1

4. Show the index after inserting an entry with hash value 68.

5. Show the original index after inserting entries with hash values 17 and 69.

6. Show the original index after deleting the entry with hash value 21. (Assume that the full deletion algorithm is used.)

7. Show the original index after deleting the entry with hash value 10. Is a merge triggered by this deletion? If not, explain why. (Assume that the full deletion algorithm is used.)

Answer 10.1 The answer to each question is given below.

1. It could be any one of the data entries in the index. We can always find a sequence of insertions and deletions with a particular key value, among the key values shown in the index as the last insertion. For example, consider the data entry 16 and the following sequence:
   1 5 21 10 15 7 51 4 12 36 64 8 24 56 16 56_D 24_D 8_D
   The last insertion is the data entry 16 and it also causes a split. But the sequence of deletions following this insertion cause a merge leading to the index structure shown in Fig 10.1.

2. The last insertion could not have caused a split because the total number of data entries in the buckets A and A_2 is 6. If the last entry caused a split the total would have been 5.

3. The last insertion which caused a split cannot be in bucket C. Buckets B and C or C and D could have made a possible bucket-split image combination but the total number of data entries in these combinations is 4 and the absence of deletions demands a sum of at least 5 data entries for such combinations. Buckets B and D can form a possible bucket-split image combination because they have a total of 6 data entries between themselves. So do A and A_2. But for the B and D to be split images the starting global depth should have been 1. If the starting global depth is 2, then the last insertion causing a split would be in A or A_2.

4. See Fig 10.2.

5. See Fig 10.3.

6. See Fig 10.4.

7. The deletion of the data entry 10 which is the only data entry in bucket C doesn't trigger a merge because bucket C is a primary page and it is left as a place holder. Right now, directory element 010 and its split image 110 already point to the same bucket C. We can't do a further merge.

   See Fig 10.5.

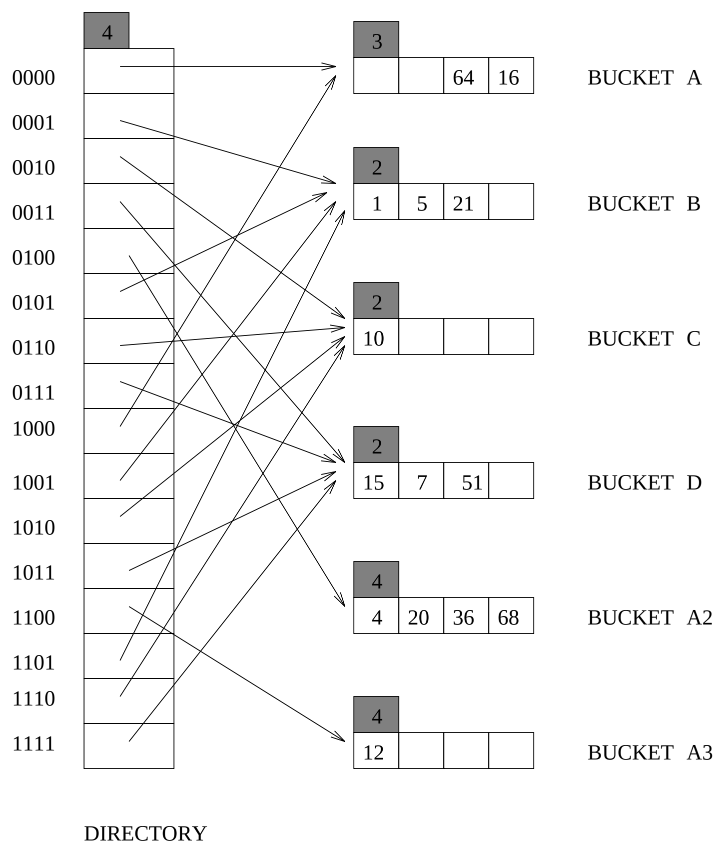
4
0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
DIRECTORY
3
64 16
BUCKET A
2
1 5 21
BUCKET B
2
10
BUCKET C
2
15 7 51
BUCKET D
4
4 20 36 68
BUCKET A2
4
12
BUCKET A3

Figure 10.2

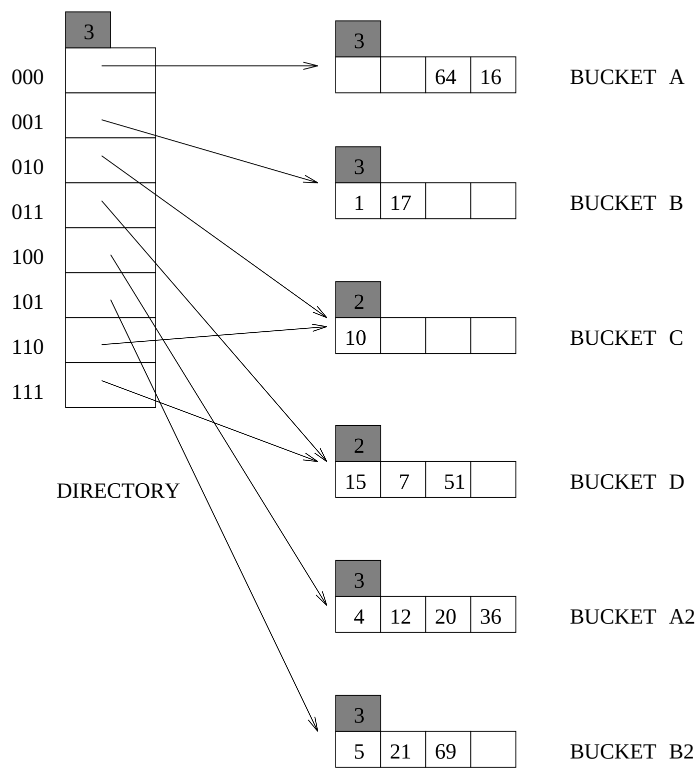
3
000
001
010
011
100
101
110
111
DIRECTORY
3
64
16
BUCKET A
3
1
17
BUCKET B
2
10
BUCKET C
2
15
7
51
BUCKET D
3
4
12
20
36
BUCKET A2
3
5
21
69
BUCKET B2

Figure 10.3

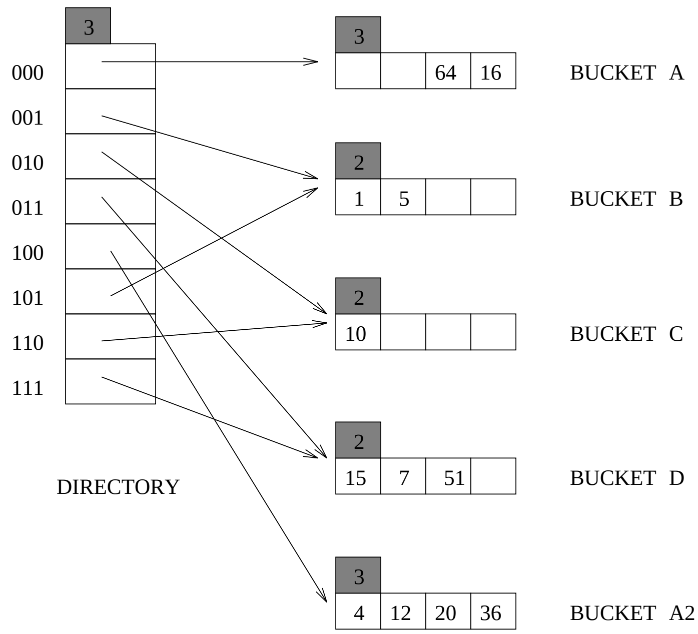

Figure 10.4

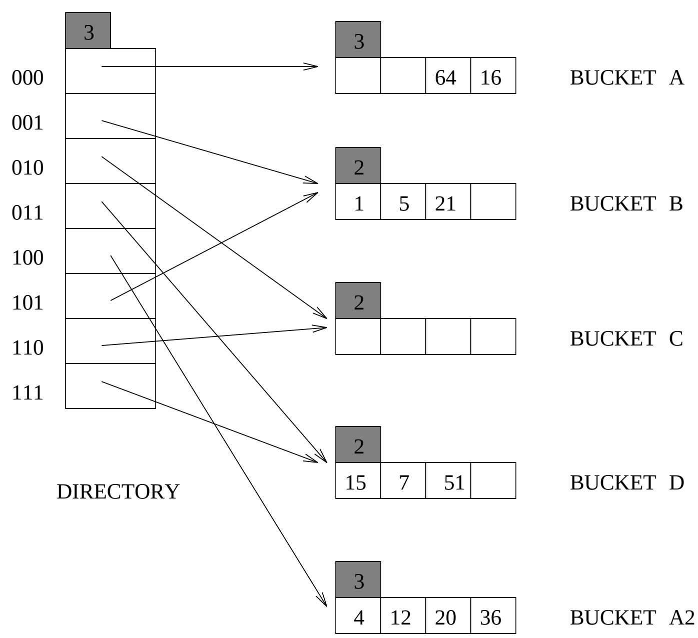

Figure 10.5

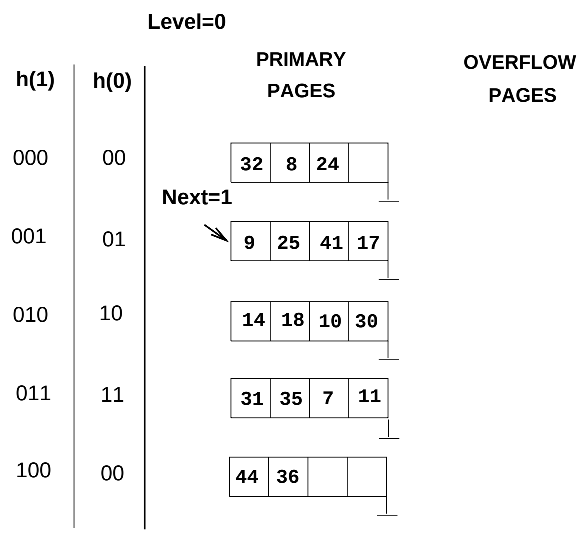

Figure 10.6 Figure for Exercise 10.2

Exercise 10.2 Consider the Linear Hashing index shown in Figure 10.6. Assume that we split whenever an overflow page is created. Answer the following questions about this index:

1. What can you say about the last entry that was inserted into the index?
2. What can you say about the last entry that was inserted into the index if you know that there have been no deletions from this index so far?
3. Suppose you know that there have been no deletions from this index so far. What can you say about the last entry whose insertion into the index caused a split?
4. Show the index after inserting an entry with hash value 4.
5. Show the original index after inserting an entry with hash value 15.
6. Show the original index after deleting the entries with hash values 36 and 44. (Assume that the full deletion algorithm is used.)
7. Find a list of entries whose insertion into the original index would lead to a bucket with two overflow pages. Use as few entries as possible to accomplish this. What is the maximum number of entries that can be inserted into this bucket before a split occurs that reduces the length of this overflow chain?

Answer 10.2 Answer omitted.

Exercise 10.3 Answer the following questions about Extendible Hashing:

1. Explain why local depth and global depth are needed.
2. After an insertion that causes the directory size to double, how many buckets have exactly one directory entry pointing to them? If an entry is then deleted from one of these buckets, what happens to the directory size? Explain your answers briefly.
3. Does Extendible Hashing guarantee at most one disk access to retrieve a record with a given key value?
4. If the hash function distributes data entries over the space of bucket numbers in a very skewed (non-uniform) way, what can you say about the size of the directory? What can you say about the space utilization in data pages (i.e., non-directory pages)?
5. Does doubling the directory require us to examine all buckets with local depth equal to global depth?
6. Why is handling duplicate key values in Extendible Hashing harder than in ISAM?

Answer 10.3 The answer to each question is given below.

1. Extendible hashing allows the size of the directory to increase and decrease depending on the number and variety of inserts and deletes. Once the directory size changes, the hash function applied to the search key value should also change. So there should be some information in the index as to which hash function is to be applied. This information is provided by the *global depth*.

   An increase in the directory size doesn't cause the creation of new buckets for each new directory entry. All the new directory entries except one share buckets with the old directory entries. Whenever a bucket which is being shared by two or more directory entries is to be split the directory size need not be doubled. This means for each bucket we need to know whether it is being shared by two or more directory entries. This information is provided by the *local depth* of the bucket. The same information can be obtained by a scan of the directory, but this is costlier.

2. Exactly two directory entries have only one directory entry pointing to them after a doubling of the directory size. This is because when the directory is doubled, one of the buckets must have split causing a directory entry to point to each of these two new buckets.

   If an entry is then deleted from one of these buckets, a merge may occur, but this depends on the deletion algorithm. If we try to merge two buckets only when a bucket becomes empty, then it is not necessary that the directory size decrease after the deletion that was considered in the question. However, if we try to merge two buckets whenever it is possible to do so then the directory size decreases after the deletion.

3. No "minimum disk access" guarantee is provided by extendible hashing. If the directory is not already in memory it needs to be fetched from the disk which may require more than one disk access depending upon the size of the directory. Then the required bucket has to be brought into the memory. Also, if alternatives 2 and 3 are followed for storing the data entries in the index then another disk access is possibly required for fetching the actual data record.

4. Consider the index in Fig 10.1. Let us consider a list of data entries with search key values of the form 2^i where $i > k$. By an appropriate choice of k, we can get all these elements mapped into the *Bucket A*. This creates 2^k elements in the directory which point to to just $k + 3$ different buckets. Also, we note there are k buckets (data pages), but just one bucket is used. So the utilization of data pages $= 1/k$.

5. No. Since we are using hashing only bucket must be examined.

6. Extendible hashing is not supposed to have overflow pages (overflow pages are dupposed to be dealt with using redistribution and splitting). When there are many duplicate entries in the index, overflow pages may be created that can never be redistributed (they will always map to the same bucket). Whenever a "split" occurs on a bucket containing only duplicate entries, an empty bucket will be created since all of the duplicates remain in the same bucket. The overflow chains will never be split, which makes inserts and searches more costly.

Exercise 10.4 Answer the following questions about Linear Hashing.

1. How does Linear Hashing provide an average-case search cost of only slightly more than one disk I/O, given that overflow buckets are part of its data structure?

2. Does Linear Hashing guarantee at most one disk access to retrieve a record with a given key value?

3. If a Linear Hashing index using Alternative (1) for data entries contains N records, with P records per page and an average storage utilization of 80 percent, what is the worst-case cost for an equality search? Under what conditions would this cost be the actual search cost?

4. If the hash function distributes data entries over the space of bucket numbers in a very skewed (non-uniform) way, what can you say about the space utilization in data pages?

Answer 10.4 Answer omitted.

Exercise 10.5 Give an example of when you would use each element (A or B) for each of the following 'A versus B' pairs:

1. A hashed index using Alternative (1) versus heap file organization.
2. Extendible Hashing versus Linear Hashing.
3. Static Hashing versus Linear Hashing.
4. Static Hashing versus ISAM.
5. Linear Hashing versus B+ trees.

Answer 10.5 The answer to each question is given below.

1. **Example 1:** Consider a situation in which most of the queries are equality queries based on the search key field. It pays to build a hashed index on this field in which case we can get the required record in one or two disk accesses. A heap file organisation may require a full scan of the file to access a particular record.
   Example 2: Consider a file on which only sequential scans are done may fare better if it is organised as a heap file. A hashed index built on it may require more disk accesses because the occupancy of the pages may not be 100%.

2. **Example 1:** Consider a set of data entries with search keys which lead to a skewed distribution of hash key values. In this case, extendible hashing causes splits of buckets at the necessary bucket whereas linear hashing goes about splitting buckets in a round-robin fashion which is useless. Here extendible hashing has a better occupancy and shorter overflow chains than linear hashing. So equality search is cheaper for extendible hashing.
   Example 2: Consider a very large file which requires a directory spanning several pages. In this case extendible hashing requires $d + 1$ disk accesses for equality selections where d is the number of directory pages. Linear hashing is cheaper.

3. **Example 1:** Consider a situation in which the number of records in the file is constant. Let all the search key values be of the form $2^n + k$ for various values of n and a few values of k. The traditional hash functions used in *linear hashing* like taking the last d bits of the search key lead to a skewed distribution of the hash key values. This leads to long overflow chains. A static hashing index can use the hash function defined as
$$h(2^n + k) = n$$
   A family of hash functions can't be built based on this hash function as k takes only a few values. In this case static hashing is better.
   Example 2: Consider a situation in which the number of records in the file varies a lot and the hash key values have a uniform distribution. Here linear hashing is clearly better than static hashing which might lead to long overflow chains thus considerably increasing the cost of equality search.

4. **Example 1:** Consider a situation in which the number of records in the file is constant and only equality selections are performed. Static hashing requires one or two disk accesses to get to the data entry. ISAM may require more than one depending on the height of the ISAM tree.
   Example 2: Consider a situation in which the search key values of data entries can be used to build a clustered index and most of the queries are range queries on this field. Then ISAM definitely wins over static hashing.

5. **Example 1:** Again consider a situation in which only equality selections are performed on the index. Linear hashing is better than B+ tree in this case.
   Example 2: When an index which is clustered and most of the queries are range searches, B+ indexes are better.

Exercise 10.6 Give examples of the following:

1. A Linear Hashing index and an Extendible Hashing index with the same data entries, such that the Linear Hashing index has more pages.
2. A Linear Hashing index and an Extendible Hashing index with the same data entries, such that the Extendible Hashing index has more pages.

Answer 10.6 Answer omitted.

Exercise 10.7 Consider a relation R(*a, b, c, d*) containing 1,000,000 records, where each page of the relation holds 10 records. R is organized as a heap file with dense secondary indexes, and the records in R are randomly ordered. Assume that attribute a is a candidate key for R, with values lying in the range 0 to 999,999. For each of the following queries, name the approach that would most likely require the fewest I/Os for processing the query. The approaches to consider follow:

- Scanning through the whole heap file for R.
- Using a B+ tree index on attribute R.a.
- Using a hash index on attribute R.a.

The queries are:

1. Find all R tuples.
2. Find all R tuples such that $a < 50$.
3. Find all R tuples such that $a = 50$.
4. Find all R tuples such that $a > 50$ and $a < 100$.

Answer 10.7 Let h be the height of the B+ tree (usually 2 or 3) and M be the number of data entries per page ($M > 10$). Let us assume that after accessing the data entry it takes one more disk access to get the actual record. Let c be the occupancy factor in hash indexing.

Consider the table shown below (disk accesses):

Problem	Heap File	B+ Tree	Hash Index
1. All tuples	10^5	$h + \frac{10^6}{M} + 10^6$	$\frac{10^6}{cM} + 10^6$
2. $a < 50$	10^5	$h + \frac{50}{M} + 50$	100
3. $a = 50$	10^5	$h + 1$	2
4. $a > 50$ and $a < 100$	10^5	$h + \frac{50}{M} + 49$	98

1. From the first row of the table, we see that heap file organization is the best (has the fewest disk accesses).
2. From the second row of the table, with typical values for h and M, the B+ Tree has the fewest disk accesses.
3. From the third row of the table, hash indexing is the best.
4. From the fourth row or the table, again we see that B+ Tree is the best.

Exercise 10.8 How would your answers to Exercise 10.7 change if attribute a is not a candidate key for R? How would they change if we assume that records in R are sorted on a?

Answer 10.8 Answer omitted.

Exercise 10.9 Consider the snapshot of the Linear Hashing index shown in Figure 10.7. Assume that a bucket split occurs whenever an overflow page is created.

1. What is the *maximum* number of data entries that can be inserted (given the best possible distribution of keys) before you have to split a bucket? Explain very briefly.
2. Show the file after inserting a *single* record whose insertion causes a bucket split.
3. (a) What is the *minimum* number of record insertions that will cause a split of all four buckets? Explain very briefly.
   (b) What is the value of *Next* after making these insertions?
   (c) What can you say about the number of pages in the fourth bucket shown after this series of record insertions?

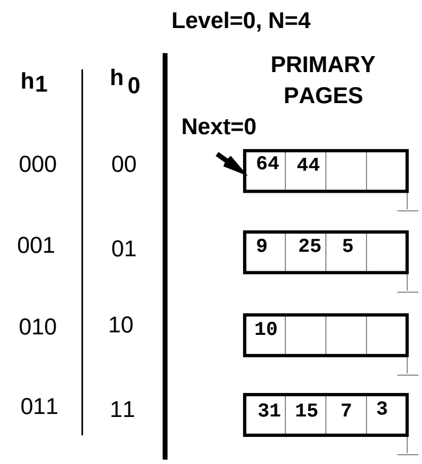

Figure 10.7 Figure for Exercise 10.9

Answer 10.9 The answer to each question is given below.

1. The maximum number of entries that can be inserted without causing a split is 6 because there is space for a total of 6 records in all the pages. A split is caused whenever an entry is inserted into a full page.

2. See Fig 10.8

3. (a) Consider the list of insertions $63, 41, 73, 137$ followed by 4 more entries which go into the same bucket, say $18, 34, 66, 130$ which go into the 3rd bucket. The insertion of 63 causes the first bucket to be split. Insertion of $41, 63$ causes the second bucket split leaving a full second bucket. Inserting 73 into it causes 3^{rd} bucket-split. At this point atleast 4 more entries are required to split the 4^{th} bucket. A minimum of 8 entries are required to cause the 4 splits.

   (b) Since all four buckets would have been split, that particular round comes to an end and the next round begins. So $Next = 0$ again.

   (c) There can be either one data page or two data pages in the fourth bucket after these insertions. If the 4 more elements inserted into the 2^{nd} bucket after 3rd bucket-spliting, then 4^{th} bucket has 1 data page.

   If the new 4 more elements inserted into the 4^{th} bucket after 4th bucket-spliting and all of them have 011 as its last three bits, then 4^{th} bucket has 2 data pages. Otherwise, if not all have 011 as its last three bits, then the 4^{th} bucket has 1 data page.

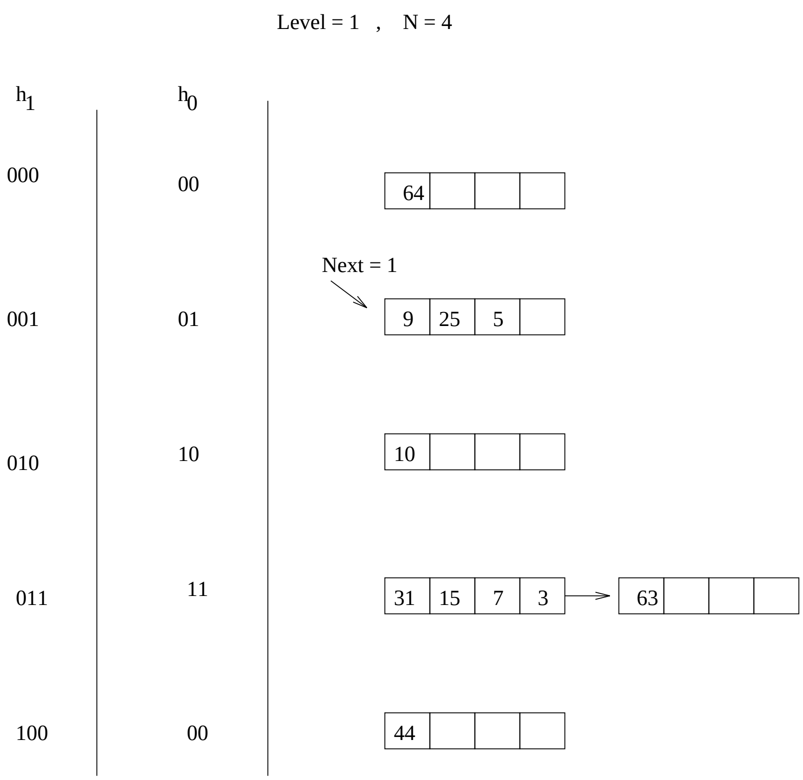
Level = 1 , N = 4
h1
h0
000
00
64
Next = 1
001
01
9
25
5
010
10
10
011
11
31
15
7
3
63
100
00
44

Figure 10.8

Exercise 10.10 Consider the data entries in the Linear Hashing index for Exercise 10.9.

1. Show an Extendible Hashing index with the same data entries.
2. Answer the questions in Exercise 10.9 with respect to this index.

Answer 10.10 Answer omitted.

Exercise 10.11 In answering the following questions, assume that the full deletion algorithm is used. Assume that merging is done when a bucket becomes empty.

1. Give an example of an Extendible Hashing index in which deleting an entry reduces the global depth.
2. Give an example of a Linear Hashing index in which deleting an entry causes *Next* to be decremented but leaves *Level* unchanged. Show the file before and after the entry is deleted.
3. Give an example of a Linear Hashing index in which deleting an entry causes *Level* to be decremented. Show the file before and after the entry is deleted.
4. Give an example of an Extendible Hashing index and a list of entries e_1, e_2, e_3 such that inserting the entries in order leads to three splits and deleting them in the reverse order yields the original index. If such an example does not exist, explain.
5. Give an example of a Linear Hashing index and a list of entries e_1, e_2, e_3 such that inserting the entries in order leads to three splits and deleting them in the reverse order yields the original index. If such an example does not exist, explain.

Answer 10.11 The answers are as follows.

1. See Fig 10.9
2. See Fig 10.10
3. See Fig 10.11
4. Let us take the transition shown in Fig 10.12. Here we insert the data entries 4, 5 and 7. Each one of these insertions causes a split with the initial split also causing a directory split. But none of these insertions redistribute the already existing data entries into the new buckets. So when we delete these data entries in the reverse order (actually the order doesn't matter) and follow the full deletion algorithm we get back the original index.

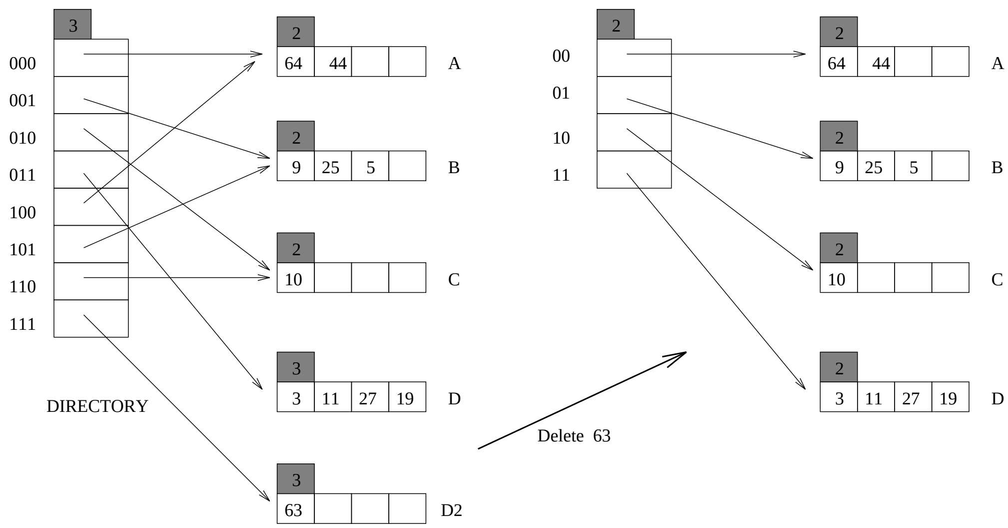

Figure 10.9

5. This example is shown in Fig 10.13. Here the idea is similar to that used in the previous answer except that the bucket being split is the one into which the insertion being made. So bucket 2 has to be split and not bucket 3. Also the order of deletions should be exactly reversed because in the deletion algorithm Next is decremented only if the last bucket becomes empty.

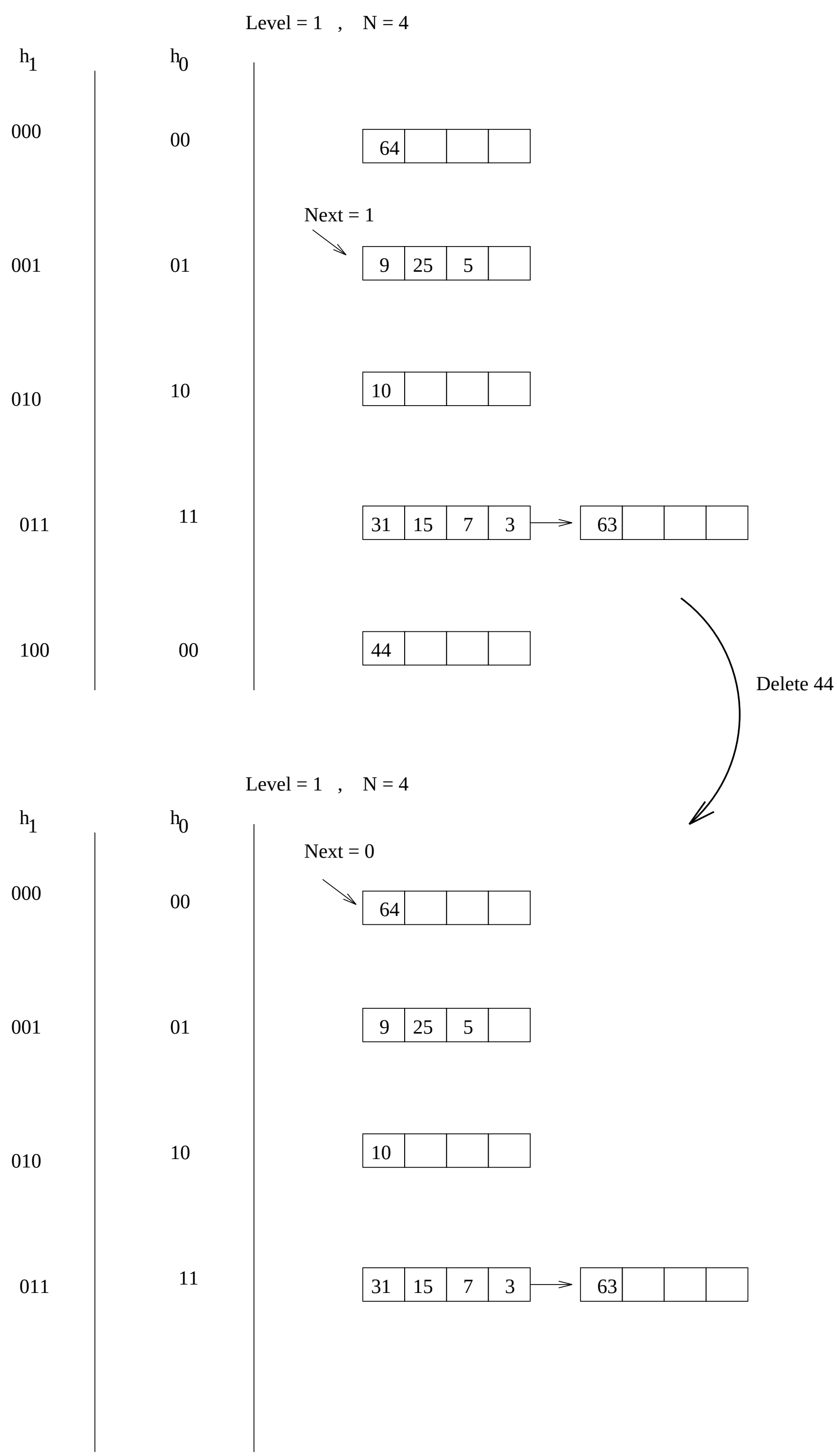
Level = 1 , N = 4
h1
h0
000
00
64
Next = 1
001
01
9
25
5
010
10
10
011
11
31
15
7
3
63
100
00
44
Delete 44
Level = 1 , N = 4
h1
h0
Next = 0
000
00
64
001
01
9
25
5
010
10
10
011
11
31
15
7
3
63

Figure 10.10

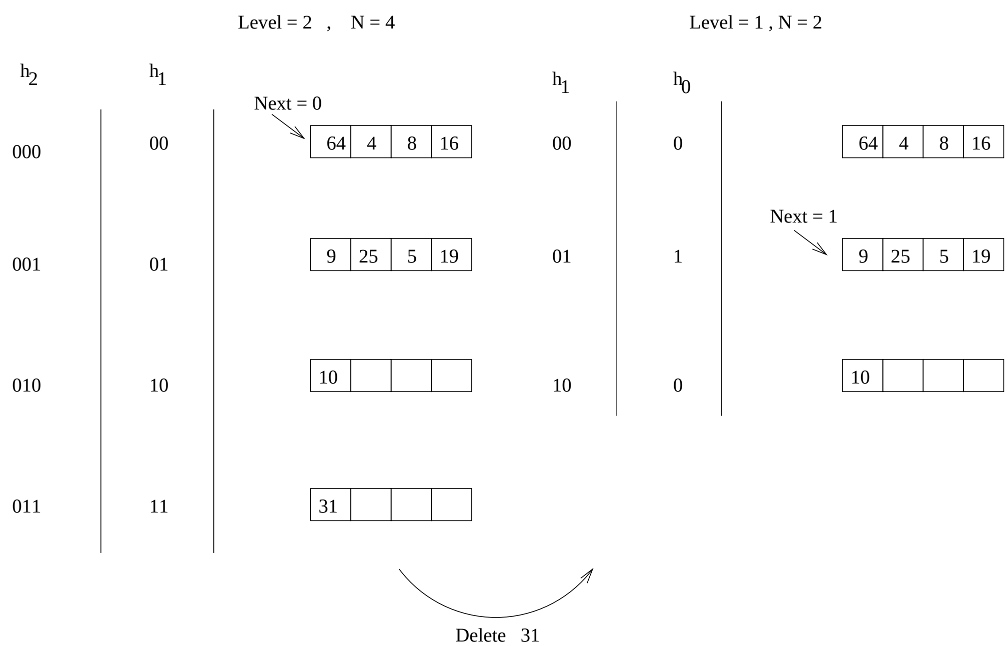
Level = 2 , N = 4
Level = 1 , N = 2
h2
h1
Next = 0
000
00
64
4
8
16
001
01
9
25
5
19
010
10
10
011
11
31
h1
h0
00
0
64
4
8
16
Next = 1
01
1
9
25
5
19
10
0
10
Delete 31

Figure 10.11

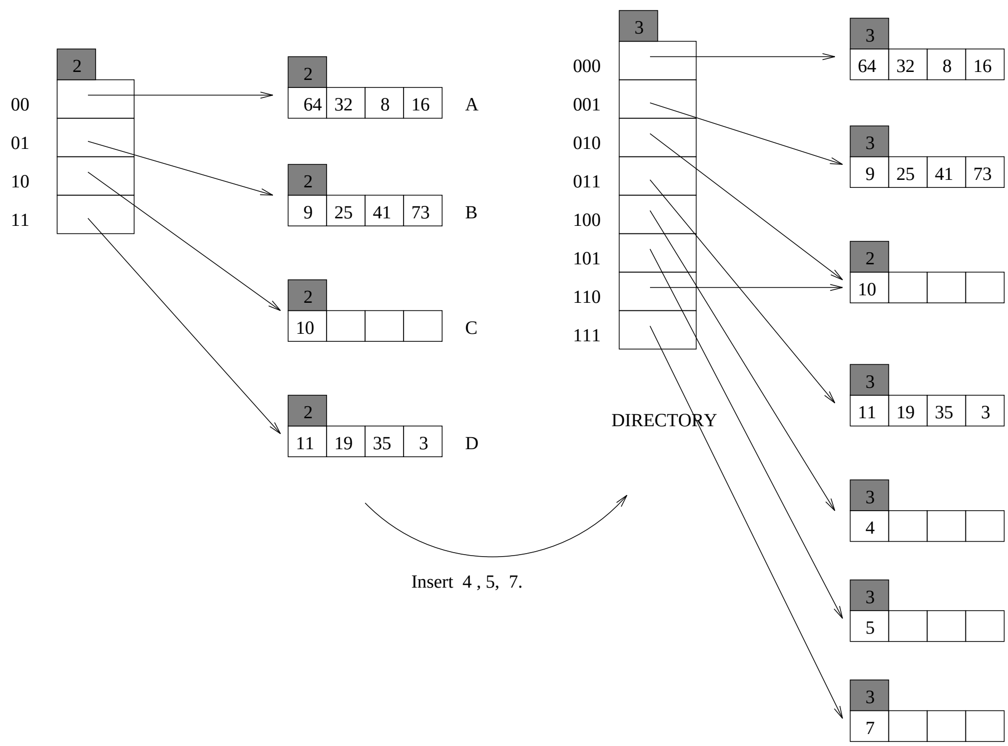
2
00
01
10
11
2
64 32 8 16
A
2
9 25 41 73
B
2
10
C
2
11 19 35 3
D
Insert 4 , 5, 7.
3
000
001
010
011
100
101
110
111
DIRECTORY
3
64 32 8 16
3
9 25 41 73
2
10
3
11 19 35 3
3
4
3
5
3
7

Figure 10.12

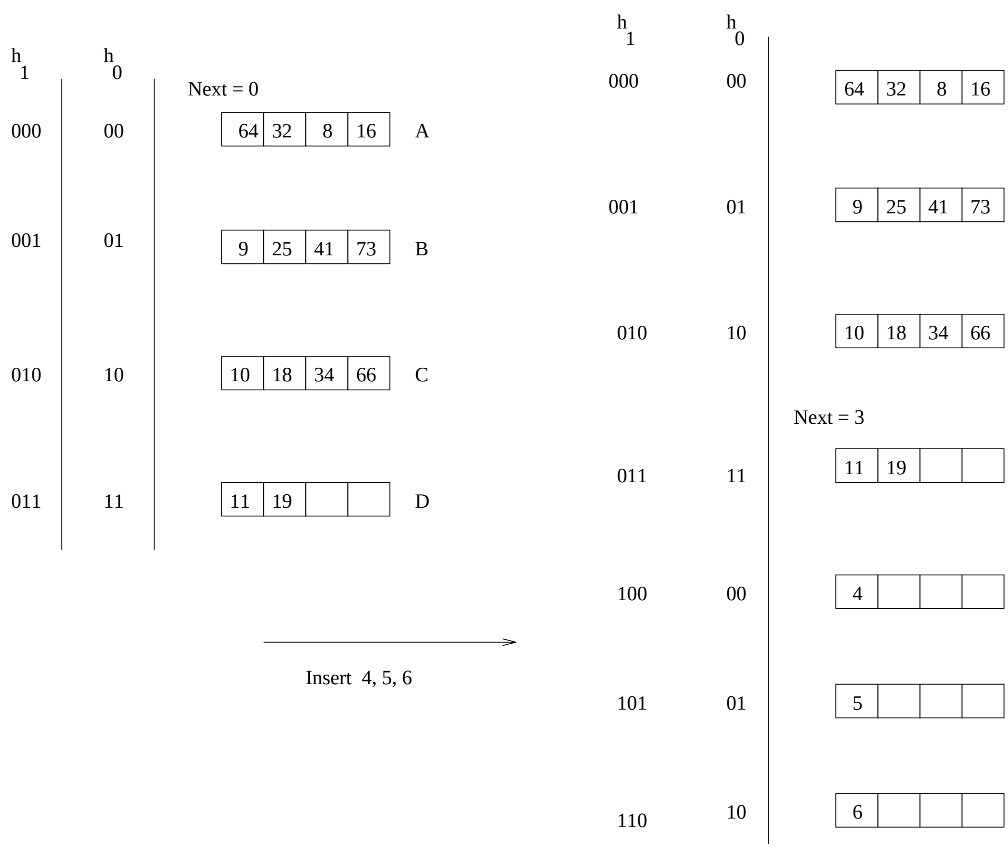
h1
h0
Next = 0
000
00
64
32
8
16
A
001
01
9
25
41
73
B
010
10
10
18
34
66
C
011
11
11
19
D
Insert 4, 5, 6
h1
h0
000
00
64
32
8
16
001
01
9
25
41
73
010
10
10
18
34
66
Next = 3
011
11
11
19
100
00
4
101
01
5
110
10
6

Figure 10.13

11

EXTERNAL SORTING

Exercise 11.1 Suppose that you have a file with 10,000 pages and that you have three buffer pages. Answer the following questions for each of these scenarios, assuming that our most general external sorting algorithm is used:

(a) A file with 10,000 pages and three available buffer pages.

(b) A file with 20,000 pages and five available buffer pages.

(c) A file with 2,000,000 pages and 17 available buffer pages.

1. How many runs will you produce in the first pass?
2. How many passes will it take to sort the file completely?
3. What is the total I/O cost of sorting the file?
4. How many buffer pages do you need to sort the file completely in just two passes?

Answer 11.1 The answer to each question is given below.

1. In the first pass (Pass 0), $\lceil N/B \rceil$ runs of B pages each are produced, where N is the number of file pages and B is the number of available buffer pages:
   (a) $\lceil 10000/3 \rceil = 3334$ sorted runs.
   (b) $\lceil 20000/5 \rceil = 4000$ sorted runs.
   (c) $\lceil 2000000/17 \rceil = 117648$ sorted runs.

2. The number of passes required to sort the file completely, including the initial sorting pass, is $\lceil log_{B-1} N1 \rceil + 1$, where $N1 = \lceil N/B \rceil$ is the number of runs produced by Pass 0:
   (a) $\lceil log_2 3334 \rceil + 1 = 13$ passes.
   (b) $\lceil log_4 4000 \rceil + 1 = 7$ passes.
   (c) $\lceil log_{16} 117648 \rceil + 1 = 6$ passes.

3. Since each page is read and written once per pass, the total number of page I/Os for sorting the file is $2 * N * (\#passes)$:
(a) 2*10000*13 = 260000.
(b) 2*20000*7 = 280000.
(c) 2*2000000*6 = 24000000.

4. In Pass 0, $\lceil N/B \rceil$ runs are produced. In Pass 1, we must be able to merge this many runs; i.e., $B - 1 \geq \lceil N/B \rceil$. This implies that B must at least be large enough to satisfy $B * (B - 1) \geq N$; this can be used to guess at B, and the guess must be validated by checking the first inequality. Thus:
(a) With 10000 pages in the file, $B = 101$ satisfies both inequalities, $B = 100$ does not, so we need 101 buffer pages.
(b) With 20000 pages in the file, $B = 142$ satisfies both inequalities, $B = 141$ does not, so we need 142 buffer pages.
(c) With 2000000 pages in the file, $B = 1415$ satisfies both inequalities, $B = 1414$ does not, so we need 1415 buffer pages.

Exercise 11.2 Answer Exercise 11.1 assuming that a two-way external sort is used.

Answer 11.2 Answer omitted.

Exercise 11.3 Suppose that you just finished inserting several records into a heap file, and now you want to sort those records. Assume that the DBMS uses external sort and makes efficient use of the available buffer space when it sorts a file. Here is some potentially useful information about the newly loaded file and the DBMS software that is available to operate on it:

> The number of records in the file is 4,500. The sort key for the file is four bytes long. You can assume that rids are eight bytes long and page ids are four bytes long. Each record is a total of 48 bytes long. The page size is 512 bytes. Each page has 12 bytes of control information on it. Four buffer pages are available.

1. How many sorted subfiles will there be after the initial pass of the sort, and how long will each subfile be?

2. How many passes (including the initial pass considered above) will be required to sort this file?

3. What will be the total I/O cost for sorting this file?

4. What is the largest file, in terms of the number of records, that you can sort with just four buffer pages in two passes? How would your answer change if you had 257 buffer pages?

5. Suppose that you have a B+ tree index with the search key being the same as the desired sort key. Find the cost of using the index to retrieve the records in sorted order for each of the following cases:

   - The index uses Alternative (1) for data entries.
   - The index uses Alternative (2) and is not clustered. (You can compute the worst-case cost in this case.)
   - How would the costs of using the index change if the file is the largest that you can sort in two passes of external sort with 257 buffer pages? Give your answer for both clustered and unclustered indexes.

Answer 11.3 The answer to each question is given below.

1. Assuming that the general external merge-sort algorithm is used, and that the available space for storing records in each page is $512 - 12 = 500$ bytes, each page can store up to 10 records of 48 bytes each. So 450 pages are needed in order to store all 4500 records, assuming that a record is not allowed to span more than one page.

   Given that 4 buffer pages are available, there will be $\lceil 450/4 \rceil = 113$ sorted runs (sub-files) of 4 pages each, except the last run, which is only 2 pages long.

2. The total number of passes will be equal to $log_3 113 + 1 = 6$ passes.

3. The total I/O cost for sorting this file is $2 * 450 * 6 = 5400$ I/Os.

4. As we saw in the previous exercise, in Pass 0, $\lceil N/B \rceil$ runs are produced. In Pass 1, we must be able to merge this many runs; i.e., $B - 1 \geq \lceil N/B \rceil$. When B is given to be 4, we get $N = 12$. The maximum number of records on 12 pages is $12 * 10 = 120$. When $B = 257$, we get $N = 65792$, and the number of records is $65792 * 10 = 657920$.

5. (a) If the index uses Alternative (1) for data entries, and it is clustered, the cost will be equal to the cost of traversing the tree from the root to the left-most leaf plus the cost of retrieving the pages in the sequence set. Assuming 67% occupancy, the number of leaf pages in the tree (the sequence set) is $450/0.67 = 600$.

   (b) If the index uses Alternative (2), and is not clustered, in the worst case, first we scan B+ tree's leaf pages, also each data entry will require fetching a data page. The number of data entries is equal to the number of data records, which is 4500. Since there is one data entry per record, each data entry requires 12 bytes, and each page holds 512 bytes, the number of B+ tree leaf pages is about $(4500 * 12)/(512 * 0.67))$, assuming 67% occupancy, which is about 150. Thus, about 4650 I/Os are required in a worst-case scenario.

(c) The B+ tree in this case has $65792/0.67 = 98197$ leaf pages if Alternative (1) is used, assuming 67% occupancy. This is the number of I/Os required (plus the relatively minor cost of going from the root to the left-most leaf). If Alternative (2) is used, and the index is not clustered, the number of I/Os is approximately equal to the number of data entries in the worst case, that is 657920,plus the number of B+ tree leaf pages 2224. Thus, number of I/Os is 660144.

Exercise 11.4 Consider a disk with an average seek time of 10ms, average rotational delay of 5ms, and a transfer time of 1ms for a 4K page. Assume that the cost of reading/writing a page is the sum of these values (i.e., 16ms) unless a *sequence* of pages is read/written. In this case the cost is the average seek time plus the average rotational delay (to find the first page in the sequence) plus 1ms per page (to transfer data). You are given 320 buffer pages and asked to sort a file with 10,000,000 pages.

1. Why is it a bad idea to use the 320 pages to support virtual memory, that is, to 'new' 10,000,000*4K bytes of memory, and to use an in-memory sorting algorithm such as Quicksort?
2. Assume that you begin by creating sorted runs of 320 pages each in the first pass. Evaluate the cost of the following approaches for the subsequent merging passes:
   (a) Do 319-way merges.
   (b) Create 256 'input' buffers of 1 page each, create an 'output' buffer of 64 pages, and do 256-way merges.
   (c) Create 16 'input' buffers of 16 pages each, create an 'output' buffer of 64 pages, and do 16-way merges.
   (d) Create eight 'input' buffers of 32 pages each, create an 'output' buffer of 64 pages, and do eight-way merges.
   (e) Create four 'input' buffers of 64 pages each, create an 'output' buffer of 64 pages, and do four-way merges.

Answer 11.4 Answer omitted.

Exercise 11.5 Consider the refinement to the external sort algorithm that produces runs of length $2B$ on average, where B is the number of buffer pages. This refinement was described in Section 11.2.1 under the assumption that all records are the same size. Explain why this assumption is required and extend the idea to cover the case of variable length records.

Answer 11.5 The assumption that all records are of the same size is used when the algorithm moves the smallest entry with a key value large than k to the output buffer and replaces it with a value from the input buffer. This "replacement" will only work if the records of the same size.

If the entries are of variable size, then we must also keep track of the size of each entry, and replace the moved entry with a new entry that fits in the available memory location. Dynamic programming algorithms have been adapted to decide an optimal replacement strategy in these cases.

12

EVALUATION OF RELATIONAL OPERATORS

Exercise 12.1 Briefly answer the following questions:

1. Consider the three basic techniques, *iteration*, *indexing*, and *partitioning*, and the relational algebra operators *selection*, *projection*, and *join*. For each technique–operator pair, describe an algorithm based on the technique for evaluating the operator.
2. Define the term *most selective access path for a query*.
3. Describe *conjunctive normal form*, and explain why it is important in the context of relational query evaluation.
4. When does a general selection condition *match* an index? What is a *primary term* in a selection condition with respect to a given index?
5. How does hybrid hash join improve upon the basic hash join algorithm?
6. Discuss the pros and cons of hash join, sort-merge join, and block nested loops join.
7. If the join condition is not equality, can you use sort-merge join? Can you use hash join? Can you use index nested loops join? Can you use block nested loops join?
8. Describe how to evaluate a grouping query with aggregation operator `MAX` using a sorting-based approach.
9. Suppose that you are building a DBMS and want to add a new aggregate operator called `SECOND LARGEST`, which is a variation of the `MAX` operator. Describe how you would implement it.
10. Give an example of how buffer replacement policies can affect the performance of a join algorithm.

Answer 12.1 The answer to each question is given below.

1. (a) *iteration–selection* Scan the entire collection, checking the condition on each tuple, and adding the tuple to the result if the condition is satisfied.

   (b) *indexing–selection* If the selection is equality and a B+ or hash index exists on the field condition, we can retrieve relevant tuples by finding them in the index and then locating them on disk.

   (c) *partitioning–selection* Do a binary search on sorted data to find the first tuple that mathes the condition. To retireve the remaining entries, we simple scan the collection starting at the first tuple we found.

   (d) *iteration–projection* Scan the entire relation, and eliminate unwanted attributes in the result.

   (e) *indexing–projection* If a multiattribute B+ tree index exists for all of the projection attributes, then one needs to only look at the leaves of the B+.

   (f) *partitioning–projection* To elimiate duplicates when doing a projection, one can simply project out the unwanted attributes and hash a combination of the remaining attributes so duplicates can be easily detected.

   (g) *iteration–join* To join two relations, one takes the first attribute in the first relation and scans the entire second relation to find tuples that match the join condition. Once the first attribute has compared to all tuples in the second relation, the second attribute from the first relation is compared to all tuples in the second relation, and so on.

   (h) *indexing–join* When an index is available, joining two relations can be more efficient. Say there are two relations A and B, and there is a secondary index on the join condition over relation A. The join works as follows: for each tuple in B, we lookup the join attribute in the index over relation A to see if there is a match. If there is a match, we store the tuple, otherwise we move to the next tuple in relation B.

   (i) *partitioning–join* One can join using partitioning by using hash join variant or a sort-merge join. For example, if there is a sort merge join, we sort both relations on the the join condition. Next, we scan both relations and identify matches. After sorting, this requires only a single scan over each relation.

2. The *most selective access path* is the query access path that retrieves the fewest pages during query evaluation. This is the most efficient way to gather the query's results.

3. $Conjuntive\ normal\ form$ is important in query evaluation because often indexes exist over some subset of conjucts in a CNF expression. Since conjuct order does not matter in CNF expressions, often indexes can be used to increase the selectivity of operators by doing a selection over two, three, or more conjucts using a single multiattribute index.

4. An index *matches* a selection condition if the index can be used to retrieve just the tuples that satisfy the condition. A *primary term* in a selection condition is a conjuct that matches an index (i.e. can be used by the index).

5. Hybrid hash join improves performance by comparing the first hash buckets during the partitioning phase rather than saving it for the probing phase. This saves us teh cost of writing and reading the first partition to disk.

6. Hash join provides excellent performance for equality joins, and can be tuned to require very few extra disk accesses beyond a one-time scan (provided enough memory is available). However, hash join is worthless for non-equality joins.

   Sort-merge joins are suitable when there is either an equality or non-equality based join condition. Sort-merge also leaves the results sorted which is often a desired property. Sort-merge join has extra costs when you have to use external sorting (there is not enough memory to do the sort in-memory).

   Block nested loops is efficient when one of the relations will fit in memory and you are using an MRU replacement strategy. However, if an index is available, there are better strategies available (but often indexes are not available).

7. If the join condition is not equality, you can use sort-merge join, index nested loops (if you have a range style index such as a B+ tree index or ISAM index), or block nested loops join. Hash joining works best for equality joins and is not suitable otherwise.

8. First we sort all of the tuples based on the `GROUP BY` attribute. Next we re-sort each group by sorting all elements on the `MAX` attribute, taking case not to re-sort beyond the group boundaries.

9. The operator `SECOND LARGEST` can be implemented using sorting. For each group (if there is a `GROUP BY` clause), we sort the tuples and return the second largest value for the desired attribute. The cost here is the cost of sorting.

10. One example where the buffer replacement stratagy effects join performance is the use of LRU and MRU in an simple nested loops join. If the relations don't fit in main memory, then the buffer strategy is critical. Say there are M buffer pages and N are filled by the first relation, and the second relation is of size M-N+P, meaning all of the second relation will fit in the the buffer except P pages. Since we must do repeated scans of the second relation, the replacement policy comes into play. With LRU, whenever we need to find a page it will have been paged out so every page request requires a disk IO. On the other hand, with MRU, we will only need to reread P-1 of the pages in the second relation, since the others will remain in memory.

Exercise 12.2 Consider a relation R(a,b,c,d,e) containing 5,000,000 records, where each data page of the relation holds 10 records. R is organized as a sorted file with dense secondary indexes. Assume that $R.a$ is a candidate key for R, with values lying in the range 0 to 4,999,999, and that R is stored in $R.a$ order. For each of the following relational algebra queries, state which of the following three approaches is most likely to be the cheapest:

- Access the sorted file for R directly.
- Use a (clustered) B+ tree index on attribute $R.a$.
- Use a linear hashed index on attribute $R.a$.

1. $\sigma_{a<50{,}000}(R)$
2. $\sigma_{a=50{,}000}(R)$
3. $\sigma_{a>50{,}000 \wedge a<50{,}010}(R)$
4. $\sigma_{a \neq 50{,}000}(R)$

Answer 12.2 Answer omitted.

Exercise 12.3 Consider processing the following SQL projection query:

```
SELECT DISTINCT E.title, E.ename FROM Executives E
```

You are given the following information:

> Executives has attributes *ename*, *title*, *dname*, and *address*; all are string fields of the same length.
> The *ename* attribute is a candidate key.
> The relation contains 10,000 pages.
> There are 10 buffer pages.

Consider the optimized version of the sorting-based projection algorithm: The initial sorting pass reads the input relation and creates sorted runs of tuples containing only attributes *ename* and *title*. Subsequent merging passes eliminate duplicates while merging the initial runs to obtain a single sorted result (as opposed to doing a separate pass to eliminate duplicates from a sorted result containing duplicates).

1. How many sorted runs are produced in the first pass? What is the average length of these runs? (Assume that memory is utilized well and that any available optimization to increase run size is used.) What is the I/O cost of this sorting pass?

2. How many additional merge passes will be required to compute the final result of the projection query? What is the I/O cost of these additional passes?

3. (a) Suppose that a clustered B+ tree index on *title* is available. Is this index likely to offer a cheaper alternative to sorting? Would your answer change if the index were unclustered? Would your answer change if the index were a hash index?

(b) Suppose that a clustered B+ tree index on *ename* is available. Is this index likely to offer a cheaper alternative to sorting? Would your answer change if the index were unclustered? Would your answer change if the index were a hash index?

(c) Suppose that a clustered B+ tree index on $\langle ename, title \rangle$ is available. Is this index likely to offer a cheaper alternative to sorting? Would your answer change if the index were unclustered? Would your answer change if the index were a hash index?

4. Suppose that the query is as follows:

   `SELECT` E.title, E.ename `FROM` Executives E

   That is, you are not required to do duplicate elimination. How would your answers to the previous questions change?

Answer 12.3 The answer to each question is given below.

1. The first pass will produce 250 sorted runs of 20 pages each, costing 15000 I/Os.

2. Using the ten buffer pages provided, on average we can write 2*10 internally sorted pages per pass, instead of 10. Then, three more passes are required to merge the 5000/20 runs, costing 2*3*5000 = 30000 I/Os.

3. (a) Using a clustered B+ tree index on *title* would reduce the cost to single scan, or 12,500 I/Os. An unclustered index could potentially cost more than 2500+100,000 (2500 from scanning the B+ tree, and 10000 * tuples per page, which I just assumed to be 10). Thus, an unclustered index would not be cheaper. Whether or not to use a hash index would depend on whether the index is clustered. If so, the hash index would probably be cheaper.

   (b) Using the clustered B+ tree on *ename* would be cheaper than sorting, in that the cost of using the B+ tree would be 12,500 I/Os. Since *ename* is a candidate key, no duplicate checking need be done for $< title, ename >$ pairs. An unclustered index would require 2500 (scan of index) + 10000 * tuples per page I/Os and thus probably be more expensive than sorting.

   (c) Using a clustered B+ tree index on $< ename, title >$ would also be more cost-effective than sorting. An unclustered B+ tree over the same attributes would allow an index-only scan, and would thus be just as economical as the clustered index. This method (both by clustered and unclustered) would cost around 5000 I/O's.

4. Knowing that duplicate elimination is not required, we can simply scan the relation and discard unwanted fields for each tuple. This is the best strategy except in the case that an index (clustered or unclustered) on $< ename, title >$ is available; in this case, we can do an index-only scan. (Note that even with `DISTINCT` specified,

no duplicates are actually present int he answer because *ename* is a candidate key. However, a typical optimizer is not likely to recognize this and omit the duplicate elimination step.)

Exercise 12.4 Consider the join $R \bowtie_{R.a=S.b} S$, given the following information about the relations to be joined. The cost metric is the number of page I/Os unless otherwise noted, and the cost of writing out the result should be uniformly ignored.

Relation R contains 10,000 tuples and has 10 tuples per page.
Relation S contains 2,000 tuples and also has 10 tuples per page.
Attribute *b* of relation S is the primary key for S.
Both relations are stored as simple heap files.
Neither relation has any indexes built on it.
52 buffer pages are available.

1. What is the cost of joining R and S using a page-oriented simple nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?
2. What is the cost of joining R and S using a block nested loops join? What is the minimum number of buffer pages required for this cost to remain unchanged?
3. What is the cost of joining R and S using a sort-merge join? What is the minimum number of buffer pages required for this cost to remain unchanged?
4. What is the cost of joining R and S using a hash join? What is the minimum number of buffer pages required for this cost to remain unchanged?
5. What would be the lowest possible I/O cost for joining R and S using *any* join algorithm, and how much buffer space would be needed to achieve this cost? Explain briefly.
6. How many tuples will the join of R and S produce, at most, and how many pages would be required to store the result of the join back on disk?
7. Would your answers to any of the previous questions in this exercise change if you are told that *R.a* is a foreign key that refers to *S.b*?

Answer 12.4 Answer omitted.

Exercise 12.5 Consider the join of R and S described in Exercise 12.1.

1. With 52 buffer pages, if unclustered B+ indexes existed on *R.a* and *S.b*, would either provide a cheaper alternative for performing the join (using an index nested loops join) than a block nested loops join? Explain.

   (a) Would your answer change if only five buffer pages were available?
   (b) Would your answer change if S contained only 10 tuples instead of 2,000 tuples?

2. With 52 buffer pages, if *clustered* B+ indexes existed on $R.a$ and $S.b$, would either provide a cheaper alternative for performing the join (using the *index nested loops* algorithm) than a block nested loops join? Explain.
   (a) Would your answer change if only five buffer pages were available?
   (b) Would your answer change if S contained only 10 tuples instead of 2,000 tuples?
3. If only 15 buffers were available, what would be the cost of a sort-merge join? What would be the cost of a hash join?
4. If the size of S were increased to also be 10,000 tuples, but only 15 buffer pages were available, what would be the cost of a sort-merge join? What would be the cost of a hash join?
5. If the size of S were increased to also be 10,000 tuples, and 52 buffer pages were available, what would be the cost of sort-merge join? What would be the cost of hash join?

Answer 12.5 Assume that it takes 3 I/Os to access a leaf in R, and 2 I/Os to access a leaf in S. And since S.b is a primary key, we will assume that every tuple in S matches 5 tuples in R.

1. The Index Nested Loops join involves probing an index on the inner relation for each tuple in the outer relation. The cost of the probe is the cost of accessing a leaf page plus the cost of retrieving any matching data records. The cost of retrieving data records could be as high as one I/O per record for an unclustered index.

   With R as the outer relation, the cost of the Index Nested Loops join will be the cost of reading R plus the cost of 10,000 probes on S.

   $$TotalCost = 1,000 + 10,000 * (2 + 1) = 31,000$$

   With S as the outer relation, the cost of the Index Nested Loops join will be the cost of reading S plus the cost of 2000 probes on R.

   $$TotalCost = 200 + 2,000 * (3 + 5) = 16,200$$

   Neither of these solutions is cheaper than Block Nested Loops join which required 4,200 I/Os.

(a) With 5 buffer pages, the cost of the Index Nested Loops joins remains the same, but the cost of the Block Nested Loops join will increase. The new cost of the Block Nested Loops join is

$$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 67,200$$

And now the cheapest solution is the Index Nested Loops join with S as the outer relation.

(b) If S contains 10 tuples then we'll need to change some of our initial assumptions. Now all of the S tuples fit on a single page, and it will only require a single I/O to access the (single) leaf in the index. Also, each tuple in S will match 1,000 tuples in R.

Block Nested Loops:

$$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 1,001$$

Index Nested Loops with R as the outer relation:

$$TotalCost = 1,000 + 10,000 * (1 + 1) = 21,000$$

Index Nested Loops with S as the outer relation:

$$TotalCost = 1 + 10 * (3 + 1,000) = 10,031$$

Block Nested Loops is still the best solution.

2. With a clustered index the cost of accessing data records becomes one I/O for every 10 data records.

With R as the outer relation, the cost of the Index Nested Loops join will be the cost of reading R plus the cost of 10,000 probes on S.

$$TotalCost = 1,000 + 10,000 * (2 + 1) = 31,000$$

With S as the outer relation, the cost of the Index Nested Loops join will be the cost of reading S plus the cost of 2000 probes on R.

$$TotalCost = 200 + 2,000 * (3 + 1) = 8,200$$

Neither of these solutions is cheaper than Block Nested Loops join which required 4,200 I/Os.

(a) With 5 buffer pages, the cost of the Index Nested Loops joins remains the same, but the cost of the Block Nested Loops join will increase. The new cost of the Block Nested Loops join is

$$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 67,200$$

And now the cheapest solution is the Index Nested Loops join with S as the outer relation.

(b) If S contains 10 tuples then we'll need to change some of our initial assumptions. Now all of the S tuples fit on a single page, and it will only require a single I/O to access the (single) leaf in the index. Also, each tuple in S will match 1,000 tuples in R.

Block Nested Loops:

$$TotalCost = N + M * \lceil \frac{N}{B-2} \rceil = 1,001$$

Index Nested Loops with R as the outer relation:

$$TotalCost = 1,000 + 10,000 * (1 + 1) = 21,000$$

Index Nested Loops with S as the outer relation:

$$TotalCost = 1 + 10 * (3 + 100) = 1,031$$

Block Nested Loops is still the best solution.

3. SORT-MERGE: With 15 buffer pages we can sort R in three passes and S in two passes. The cost of sorting R is $2 * 3 * M = 6,000$, the cost of sorting S is $2 * 2 * N = 800$, and the cost of the merging phase is $M + N = 1,200$.

$$TotalCost = 6,000 + 800 + 1,200 = 8,000$$

HASH JOIN: With 15 buffer pages the first scan of S (the smaller relation) splits it into 14 buckets, each containing about 15 pages. To store one of these buckets (and its hash table) in memory will require $f * 15$ pages, which is more than we have available. We must apply the Hash Join technique again to all partitions of R and S that were created by the first partitioning phase. Then we can fit an entire partition of S in memory. The total cost will be the cost of two partioning phases plus the cost of one matching phase.

$$TotalCost = 2 * (2 * (M + N)) + (M + N) = 6,000$$

4. SORT-MERGE: With 15 buffer pages we can sort R in three passes and S in three passes. The cost of sorting R is $2 * 3 * M = 6,000$, the cost of sorting S is $2 * 3 * N = 6,000$, and the cost of the merging phase is $M + N = 2,000$.

$$TotalCost = 6,000 + 6,000 + 2,000 = 14,000$$

HASH JOIN: Now both relations are the same size, so we can treat either one as the smaller relation. With 15 buffer pages the first scan of S splits it into 14 buckets, each containing about 72 pages, so again we have to deal with partition overflow. We must apply the Hash Join technique again to all partitions of R and S that were created by the first partitioning phase. Then we can fit an entire partition of S in memory. The total cost will be the cost of two partioning phases plus the cost of one matching phase.

$$TotalCost = 2 * (2 * (M + N)) + (M + N) = 10,000$$

5. SORT-MERGE: With 52 buffer pages we have $B > \sqrt{M}$ so we can use the "merge-on-the-fly" refinement which costs $3 * (M + N)$.

$$TotalCost = 3 * (1,000 + 1,000) = 6,000$$

HASH JOIN: Now both relations are the same size, so we can treat either one as the smaller relation. With 52 buffer pages the first scan of S splits it into 51 buckets, each containing about 20 pages. This time we do not have to deal with partition overflow. The total cost will be the cost of one partioning phase plus the cost of one matching phase.

$$TotalCost = (2 * (M + N)) + (M + N) = 6,000$$

Exercise 12.6 Answer each of the questions—if some question is inapplicable, explain why—in Exercise 12.1 again, but using the following information about R and S:

Relation R contains 200,000 tuples and has 20 tuples per page.
Relation S contains 4,000,000 tuples and also has 20 tuples per page.
Attribute a of relation R is the primary key for R.
Each tuple of R joins with exactly 20 tuples of S.
1,002 buffer pages are available.

Answer 12.6 Answer omitted.

Exercise 12.7 We described variations of the join operation called *outer joins* in Section 5.6.4. One approach to implementing an outer join operation is to first evaluate the corresponding (inner) join and then add additional tuples padded with *null* values to the result in accordance with the semantics of the given outer join operator. However, this requires us to compare the result of the inner join with the input relations to determine the additional tuples to be added. The cost of this comparison can be avoided by modifying the join algorithm to add these extra tuples to the result while input tuples are processed during the join. Consider the following join algorithms: *block nested loops join, index nested loops join, sort-merge join,* and *hash join*. Describe how you would modify each of these algorithms to compute the following operations on the Sailors and Reserves tables discussed in this chapter:

1. Sailors `NATURAL LEFT OUTER JOIN` Reserves
2. Sailors `NATURAL RIGHT OUTER JOIN` Reserves
3. Sailors `NATURAL FULL OUTER JOIN` Reserves

Answer 12.7 Each join method is considered in turn.

1. Sailors (S) `NATURAL LEFT OUTER JOIN` Reserves (R)
   In this `LEFT OUTER JOIN`, Sailor rows without a matching Reserves row will appear in the result with a *null* value for the Reserves value.

   (a) *block nested loops join*
   In the *block nested loops join* algorithm, we place as large a partition of the Sailors relation in memory as possibly, leaving 2 extra buffer pages (one for input pages of R, the other for output pages plus enough pages for a single bit for each record of the block of S. These 'bit pages' are initially set to zero; when a tuple of R matches a tuple in S, the bit is set to 1 meaning that this page has already met the join condition. Once all of R has been compared to the block of S, any tuple with its bit still set to zero is added to the ouput with a *null* value for the R tuple. This process is then repeated for the remaining blocks of S.

   (b) *index nested loops join*
   An *index nested loops join* requires an index for Reserves on all attributes that Sailors and Reserves have in common. For each tuple in Sailors, if it matches a tuple in the R index, it is added to the output, otherwise the S tuple is added to the output with a *null* value.

   (c) *sort-merge join*
   When the two relations are merged, Sailors is scanned in sorted order and if there is no match in Reserves, the Sailors tuple is added to the output with a *null* value.

   (d) *hash join*
   We hash so that partitions of Reserves will fit in memory with enough leftover space to hold a page of the corresponding Sailors partition. When we compare a Sailors tuple to all of the tuples in the Reserves partition, if there is a match it is added to the output, otherwise we add the S tuple and a *null* value to the output.

2. Sailors `NATURAL RIGHT OUTER JOIN` Reserves
   In this `RIGHT OUTER JOIN`, Reserves rows without a matching Sailors row will appear in the result with a *null* value for the Sailors value.

   (a) *block nested loops join*
   In the *block nested loops join* algorithm, we place as large a partition of the Reserves relation in memory as possibly, leaving 2 extra buffer pages (one for input pages of Sailors, the other for output pages plus enough pages for a single bit for each record of the block of R. These 'bit pages' are initially set to zero; when a tuple of S matches a tuple in R, the bit is set to 1 meaning that this page has already met the join condition. Once all of S has been compared to the block of R, any tuple with its bit still set to zero is added to the ouput with a *null* value for the S tuple. This process is then repeated for the remaining blocks of R.

(b) *index nested loops join*
An *index nested loops join* requires an index for Sailors on all attributes that Reserves and Sailors have in common. For each tuple in Reserves, if it matches a tuple in the S index, it is added to the output, otherwise the R tuple is added to the output with a *null* value.

(c) *sort-merge join*
When the two relations are merged, Reserves is scanned in sorted order and if there is no match in Sailors, the Reserves tuple is added to the output with a *null* value.

(d) *hash join*
We hash so that partitions of Sailors will fit in memory with enough leftover space to hold a page of the corresponding Reserves partition. When we compare a Reserves tuple to all of the tuples in the Sailors partition, if there is a match it is added to the output, otherwise we add the Reserves tuple and a *null* value to the output.

3. Sailors `NATURAL FULL OUTER JOIN` Reserves
In this `FULL OUTER JOIN`, Sailor rows without a matching Reserves row will appear in the result with a *null* value for the Reserves value, and Reserves rows without a matching Sailors row will appear in teh result with a *null* value.

(a) *block nested loops join*
For this algorithm to work properly, we need a bit for each tuple in both relations. If after completing the join there are any bits still set to zero, these tuples are joined with *null* values.

(b) *index nested loops join*
If there is only an index on one relation, we can use that index to find half of the full outer join in a similar fashion as in the `LEFT` and `RIGHT OUTER` joins. To find the non-matches of the relation with the index, we can use the same trick as in the *block nested loops join* and keep bit flags for each block of scans.

(c) *sort-merge join*
During the merge phase, we scan both relations alternating to the relation with the lower value. If that tuple has no match, it is added to the output with a *null* value.

(d) *hash join*
When we hash both relations, we should choose a hash function that will hash the larger relation into partitions that will fit in half of memory. This way we can fit both relations' partitions into main memory and we can scan both relations for matches. If no match is found (we must scan for both relations), then we add that tuple to the output with a *null* value.

13

INTRODUCTION TO QUERY OPTIMIZATION

Exercise 13.1 Briefly answer the following questions.

1. What is the goal of query optimization? Why is it important?
2. Describe the advantages of *pipelining*.
3. Give an example in which pipelining *cannot* be used.
4. Describe the *iterator* interface and explain its advantages.
5. What role do statistics gathered from the database play in query optimization?
6. What information is stored in the system catalogs?
7. What are the benefits of making the system catalogs be relations?
8. What were the important design decisions made in the System R optimizer?

Answer 13.1

1. The goal of query optimization is to avoid the worst plans and find a good plan. The goal is usually not to find the optimal plan. The difference in cost between a good plan and a bad plan can be several orders of magnitude: a good query plan can evaluate the query in seconds, whereas a bad query plan might take days!
2. Pipelining allows us to avoid creating and reading temporary relations; the I/O savings can be substantial.
3. Bushy query plans often cannot take advantage of pipelining because of limited buffer or CPU resources. Consider a bushy plan in which we are doing a selection on two relations, followed by a join. We cannot always use pipelining in this strategy becuase the result of the selection on the first selection may not fit in memory, and we must wait for the second relation's selection to complete before we can begin the join.

4. The iterator interface for an operator includes the functions *open*, *get_next*, and *close*; it hides the details of how the operator is implemented, and allows us to view all operator nodes in a query plan uniformly.

5. The query optimizer uses statistics to improve the chances of selecting an optimum query plan. The statistics are used to calculate reduction factors which determine the results the optimizer may expect given different indexes and inputs.

6. Information about relations, indexes, and views is stored in the system catalogs. This includes file names, file sizes, and file structure, the attribute names and data types, lists of keys, and constraints.

   Some commonly stored statistical information includes:

   (a) Cardinality - the number of tuples for each relation
   (b) Size - the number of pages in each relation
   (c) Index Cardinality - the number of distinct key values for each index
   (d) Index Size - the number of pages for each index (or number of leaf pages)
   (e) Index Height - the number of nonleaf levels for each tree index
   (f) Index Range - the minimum present key value and the maximum present key value for each index.

7. There are several advantages to storing the system catatlogs as relations. Relational system catalogs take advantage of all of the implementation and management benefits of relational tables: effective information storage and rich querying capabilities. The choice of what system catalogs to maintain is left to the DBMS implementor.

8. Some important design decisions in the System R optimizer are:

   (a) Using statistics about a database instance to estimate the cost of a query evaluation plan.
   (b) A decision to consider only plans with binary joins in which the inner plan is a base relation. This heuristic reduces the often significant number of alternative plans that must be considered.
   (c) A decision to focus optimization on the class of SQL queries without nesting and to treat nested queries in a relatively ad hoc way.
   (d) A decision not to perform duplicate elimination for projections (except as a final step in the query evaluation when required by a `DISTINCT` clause).
   (e) A model of cost that accounted for CPU costs as wellas I/O costs.

14

A TYPICAL QUERY OPTIMIZER

Exercise 14.1 Briefly answer the following questions.

1. In the context of query optimization, what is an *SQL query block*?
2. Define the term *reduction factor*.
3. Describe a situation in which projection should precede selection in processing a project-select query, and describe a situation where the opposite processing order is better. (Assume that duplicate elimination for projection is done via sorting.)
4. If there are dense, unclustered (secondary) B+ tree indexes on both $R.a$ and $S.b$, the join $R \bowtie_{a=b} S$ could be processed by doing a sort-merge type of join—without doing any sorting—by using these indexes.
   (a) Would this be a good idea if R and S each have only one tuple per page, or would it be better to ignore the indexes and sort R and S? Explain.
   (b) What if R and S each have many tuples per page? Again, explain.
5. Why does the System R optimizer consider only left-deep join trees? Give an example of a plan that would not be considered because of this restriction.
6. Explain the role of *interesting orders* in the System R optimizer.

Answer 14.1 The answer to each question is given below.

1. An SQL query block is an SQL query without nesting, and serves as a unit of optimization. Blocks have one `SELECT` statement, one `FROM` statement, and at most one `WHERE`, one `GROUP BY`, and one `HAVING` statements. Queries with nesting can be broken up into a collection of query blocks whose evaluation must be coordinated at runtime.
2. The *reduction factor* for a term, is the ratio between the expected result size to the input size, considering only the slection represented by the term.

3. If the selection is to be done on the inner relation of a simple nested loop, and the projection will reduce the number of pages occupied significantly, then the projection should be done first.

   The opposite is true in the case of an index-only join. The projections should be done on the fly after the join.

4. (a) Using the indexes is a good idea when R and S each have only one tuple per page. Each data page is read exactly once and the cost of scanning the B+ tree is likely to be very small.

   (b) Doing an actual data sort on appropriate keys may be a good idea when R and S have many tuples per page. Given that the indexes are unclustered, without sorting there is potential for many reads of a single page. After sorting, there will only be one read per matching page. The choice may be determined by number of potential matches and number of tuples per page.

5. The System-R optimizer considers only left-deep joins because they allow fully pipelined plans. As an example, non-linear plans would not be considered.

6. The System R optimizer implements a multiple pass algorithm. In each pass, it must consider adding a join to those retained in previous passes. Each level retains the cheapest plan for each interesting order for result tuples. An ordering of tuples is interesting if it is sorted on some combination of fields.

Exercise 14.2 Consider a relation with this schema:

Employees($\underline{eid\text{: }\texttt{integer}}$, *ename:* `string`, *sal:* `integer`, *title:* `string`, *age:* `integer`)

Suppose that the following indexes, all using Alternative (2) for data entries, exist: a hash index on *eid*, a B+ tree index on *sal*, a hash index on *age*, and a clustered B+ tree index on $\langle age, sal \rangle$. Each Employees record is 100 bytes long, and you can assume that each index data entry is 20 bytes long. The Employees relation contains 10,000 pages.

1. Consider each of the following selection conditions and, assuming that the reduction factor (RF) for each term that matches an index is 0.1, compute the cost of the most selective access path for retrieving all Employees tuples that satisfy the condition:

   (a) $sal > 100$

   (b) $age = 25$

   (c) $age > 20$

   (d) $eid = 1,000$

   (e) $sal > 200 \wedge age > 30$

(f) $sal > 200 \wedge age = 20$

(g) $sal > 200 \wedge title =' CFO'$

(h) $sal > 200 \wedge age > 30 \wedge title =' CFO'$

2. Suppose that for each of the preceding selection conditions, you want to retrieve the average salary of qualifying tuples. For each selection condition, describe the least expensive evaluation method and state its cost.

3. Suppose that for each of the preceding selection conditions, you want to compute the average salary for each *age* group. For each selection condition, describe the least expensive evaluation method and state its cost.

4. Suppose that for each of the preceding selection conditions, you want to compute the average age for each *sal* level (i.e., group by *sal*). For each selection condition, describe the least expensive evaluation method and state its cost.

5. For each of the following selection conditions, describe the best evaluation method:

   (a) $sal > 200 \vee age = 20$

   (b) $sal > 200 \vee title =' CFO'$

   (c) $title =' CFO' \wedge ename =' Joe'$

Answer 14.2 Answer omitted.

Exercise 14.3 For each of the following SQL queries, for each relation involved, list the attributes that must be examined in order to compute the answer. All queries refer to the following relations:

Emp(*eid:* `integer`, *did:* `integer`, *sal:* `integer`, *hobby:* `char(20)`)
Dept(*did:* `integer`, *dname:* `char(20)`, *floor:* `integer`, *budget:* `real`)

1. `SELECT` * `FROM` Emp

2. `SELECT` * `FROM` Emp, Dept

3. `SELECT` * `FROM` Emp E, Dept D `WHERE` E.did = D.did

4. `SELECT` E.eid, D.dname `FROM` Emp E, Dept D `WHERE` E.did = D.did

5. `SELECT` `COUNT`(*) `FROM` Emp E, Dept D `WHERE` E.did = D.did

6. `SELECT` `MAX`(E.sal) `FROM` Emp E, Dept D `WHERE` E.did = D.did

7. `SELECT` `MAX`(E.sal) `FROM` Emp E, Dept D `WHERE` E.did = D.did `AND` D.floor = 5

8. `SELECT` E.did, `COUNT(*)` `FROM` Emp E, Dept D `WHERE` E.did = D.did `GROUP BY` D.did

9. `SELECT` D.floor, `AVG`(D.budget) `FROM` Dept D `GROUP BY` D.floor `HAVING COUNT(*)` > 2

10. `SELECT` D.floor, `AVG`(D.budget) `FROM` Dept D `GROUP BY` D.floor `ORDER BY` D.floor

Answer 14.3 The answer to each question is given below.

1. E.eid, E.did, E.sal, E.hobby
2. E.eid, E.did, E.sal, E.hobby, D.did, D.dname, D.floor, D.budget
3. E.eid, E.did, E.sal, E.hobby, D.did, D.dname, D.floor, D.budget
4. E.eid, D.dname, E.did, D.did
5. E.did, D.did
6. E.sal, E.did, D.did
7. E.sal, E.did, D.did, D.floor
8. E.did, D.did
9. D.floor, D.budget
10. D.floor, D.budget

Exercise 14.4 You are given the following information:

> Executives has attributes *ename*, *title*, *dname*, and *address*; all are string fields of the same length.
> The *ename* attribute is a candidate key.
> The relation contains 10,000 pages.
> There are 10 buffer pages.

1. Consider the following query:

   `SELECT` E.title, E.ename `FROM` Executives E `WHERE` E.title='CFO'

   Assume that only 10 percent of Executives tuples meet the selection condition.

   (a) Suppose that a clustered B+ tree index on *title* is (the only index) available. What is the cost of the best plan? (In this and subsequent questions, be sure to describe the plan that you have in mind.)

(b) Suppose that an unclustered B+ tree index on *title* is (the only index) available. What is the cost of the best plan?

(c) Suppose that a clustered B+ tree index on *ename* is (the only index) available. What is the cost of the best plan?

(d) Suppose that a clustered B+ tree index on *address* is (the only index) available. What is the cost of the best plan?

(e) Suppose that a clustered B+ tree index on $\langle ename, title \rangle$ is (the only index) available. What is the cost of the best plan?

2. Suppose that the query is as follows:

```
SELECT E.ename FROM Executives E WHERE E.title='CFO' AND E.dname='Toy'
```

Assume that only 10 percent of Executives tuples meet the condition $E.title =' CFO'$, only 10 percent meet $E.dname =' Toy'$, and that only 5 percent meet both conditions.

(a) Suppose that a clustered B+ tree index on *title* is (the only index) available. What is the cost of the best plan?

(b) Suppose that a clustered B+ tree index on *dname* is (the only index) available. What is the cost of the best plan?

(c) Suppose that a clustered B+ tree index on $\langle title, dname \rangle$ is (the only index) available. What is the cost of the best plan?

(d) Suppose that a clustered B+ tree index on $\langle title, ename \rangle$ is (the only index) available. What is the cost of the best plan?

(e) Suppose that a clustered B+ tree index on $\langle dname, title, ename \rangle$ is (the only index) available. What is the cost of the best plan?

(f) Suppose that a clustered B+ tree index on $\langle ename, title, dname \rangle$ is (the only index) available. What is the cost of the best plan?

3. Suppose that the query is as follows:

```
SELECT E.title, COUNT(*) FROM Executives E GROUP BY E.title
```

(a) Suppose that a clustered B+ tree index on *title* is (the only index) available. What is the cost of the best plan?

(b) Suppose that an unclustered B+ tree index on *title* is (the only index) available. What is the cost of the best plan?

(c) Suppose that a clustered B+ tree index on *ename* is (the only index) available. What is the cost of the best plan?

(d) Suppose that a clustered B+ tree index on $\langle ename, title \rangle$ is (the only index) available. What is the cost of the best plan?

   (e) Suppose that a clustered B+ tree index on $\langle title, ename \rangle$ is (the only index) available. What is the cost of the best plan?

4. Suppose that the query is as follows:

   ```
   SELECT E.title, COUNT(*) FROM Executives E
   WHERE E.dname > 'W%' GROUP BY E.title
   ```

   Assume that only 10 percent of Executives tuples meet the selection condition.

   (a) Suppose that a clustered B+ tree index on *title* is (the only index) available. What is the cost of the best plan? If an additional index (on any search key that you want) is available, would it help to produce a better plan?

   (b) Suppose that an unclustered B+ tree index on *title* is (the only index) available. What is the cost of the best plan?

   (c) Suppose that a clustered B+ tree index on *dname* is (the only index) available. What is the cost of the best plan? If an additional index (on any search key that you want) is available, would it help to produce a better plan?

   (d) Suppose that a clustered B+ tree index on $\langle dname, title \rangle$ is (the only index) available. What is the cost of the best plan?

   (e) Suppose that a clustered B+ tree index on $\langle title, dname \rangle$ is (the only index) available. What is the cost of the best plan?

Answer 14.4 Answer omitted.

Exercise 14.5 Consider the query $\pi_{A,B,C,D}(R \bowtie_{A=C} S)$. Suppose that the projection routine is based on sorting and is smart enough to eliminate all but the desired attributes during the initial pass of the sort, and also to toss out duplicate tuples on-the-fly while sorting, thus eliminating two potential extra passes. Finally, assume that you know the following:

R is 10 pages long, and R tuples are 300 bytes long.
S is 100 pages long, and S tuples are 500 bytes long.
C is a key for S, and A is a key for R.
The page size is 1,024 bytes.
Each S tuple joins with exactly one R tuple.
The combined size of attributes A, B, C, and D is 450 bytes.
A and B are in R and have a combined size of 200 bytes; C and D are in S.

1. What is the cost of writing out the final result? (As usual, you should ignore this cost in answering subsequent questions.)

2. Suppose that three buffer pages are available, and the only join method that is implemented is simple (page-oriented) nested loops.

(a) Compute the cost of doing the projection followed by the join.

(b) Compute the cost of doing the join followed by the projection.

(c) Compute the cost of doing the join first and then the projection on-the-fly.

(d) Would your answers change if 11 buffer pages were available?

3. Suppose that there are three buffer pages available, and the only join method that is implemented is block nested loops.

   (a) Compute the cost of doing the projection followed by the join.

   (b) Compute the cost of doing the join followed by the projection.

   (c) Compute the cost of doing the join first and then the projection on-the-fly.

   (d) Would your answers change if 11 buffer pages were available?

Answer 14.5 The answer to each question is given below.

1. From the given information, we know that R has 30 tuples (10 pages of 3 records each), and S has 200 tuples (100 pages of 2 records each). Since every S tuple joins with exaclty one R tuple, there can be at most 200 tuples after the join. Since the size of the result is 450 bytes/record, 2 records will fit on a page. This means 200 / 2 = 100 page writes are needed to write the result to disk.

2. (a) Cost of projection followed by join: The projection is sort-based, so we must sort relation S, which contains attributes C and D. Relation S has 100 pages, and we have 3 buffer pages, so the sort cost is $200 * \lceil log2(100) \rceil = 200 * 7 = 1400$.

   Assume that 1/10 of the tuples are removed as duplicates, so that there are 180 remaining tuples of S, each of size 150 bytes (combined size of attributes C, D). Therefore, 6 tuples fit on a page, so the resulting size of the inner relation is 30 pages.

   The projection on R is calculated similarly: R has 10 pages, so the sort will cost $30 * \lceil log2(10) \rceil = 30 * 4 = 120$. If 1/10 are not duplicates, then there are 27 tuples remaining, each of size 200 bytes. Therefore, 5 tuples fit on a page so the resulting size of the outer relation is 6 pages.

   The cost using SNL is (6 + 6*30) = 186 I/Os, for a total cost of 1586 I/Os.

   (b) Cost of join followed by projection:

   SNL join is (10 + 10*100) = 1010 I/Os, and results in 200 tuples, each of size 800 bytes. Thus, only one result tuple fits on a page, and we have 200 pages.

   The projection is a sort using 3 buffer pages, and in the first pass unwanted attributes are eliminated on-the-fly to produce tuples of size 450 bytes, i.e., 2 tuples per page. Thus, 200 pages are scanned and 100 pages written in the first pass in 33 runs of 3 pages each and 1 run of a page. These runs are merged pairwise in 6 additional passes for a total projection cost of 200+100+2*6*100=1500 I/Os. This includes the cost of writing out the result of 100 pages; removing this cost and adding the cost of the join step, we obtain a total cost of 2410 I/Os.

(c) Cost of join and projection on the fly:

This means that the projection cost is 0, so the only cost is the join, which we know from above is 1010 I/Os.

(d) If we had 11 buffer pages, then the projection sort could be done log10 instead of log2.

3. (a) Since BNL with 3 pages is just SNL (page oriented), and the projection sorts are the same as well (same buffer size), the cost is the same as in 2a.

   (b) Cost of join and then projection: Same as 2b.

   (c) Cost of join and projection on the fly Same as 2c

   (d) Now that we have 11 buffer pages, both BNL and projections are affected.

      Part (a): $log10(100) * 200 + (10 + \lceil 10/9 \rceil * 45) = 500$
      Part (b): $(10 + \lceil 10/9 \rceil * 100) + (\lceil log10(200) \rceil * 400) = 210 + 1200 = 1410$
      Part (c): $10 + \lceil 10/9 \rceil * 100 = 210$

Exercise 14.6 Briefly answer the following questions.

1. Explain the role of relational algebra equivalences in the System R optimizer.

2. Consider a relational algebra expression of the form $\sigma_c(\pi_l(R \times S))$. Suppose that the equivalent expression with selections and projections pushed as much as possible, taking into account only relational algebra equivalences, is in one of the following forms. In each case give an illustrative example of the selection conditions and the projection lists (c, l, $c1$, $l1$, etc.).

   (a) *Equivalent maximally pushed form:* $\pi_{l1}(\sigma_{c1}(R) \times S)$.

   (b) *Equivalent maximally pushed form:* $\pi_{l1}(\sigma_{c1}(R) \times \sigma_{c2}(S))$.

   (c) *Equivalent maximally pushed form:* $\sigma_c(\pi_{l1}(\pi_{l2}(R) \times S))$.

   (d) *Equivalent maximally pushed form:* $\sigma_{c1}(\pi_{l1}(\sigma_{c2}(\pi_{l2}(R)) \times S))$.

   (e) *Equivalent maximally pushed form:* $\sigma_{c1}(\pi_{l1}(\pi_{l2}(\sigma_{c2}(R)) \times S))$.

   (f) *Equivalent maximally pushed form:* $\pi_l(\sigma_{c1}(\pi_{l1}(\pi_{l2}(\sigma_{c2}(R)) \times S)))$.

Answer 14.6 Answer omitted.

Exercise 14.7 Consider the following relational schema and SQL query. The schema captures information about employees, departments, and company finances (organized on a per department basis).

Emp(*eid:* `integer`, *did:* `integer`, *sal:* `integer`, *hobby:* `char(20)`)
Dept(*did:* `integer`, *dname:* `char(20)`, *floor:* `integer`, *phone:* `char(10)`)
Finance(*did:* `integer`, *budget:* `real`, *sales:* `real`, *expenses:* `real`)

Consider the following query:

```
SELECT D.dname, F.budget
FROM   Emp E, Dept D, Finance F
WHERE  E.did=D.did AND D.did=F.did AND D.floor=1
       AND E.sal ≥ 59000 AND E.hobby = 'yodeling'
```

1. Identify a relational algebra tree (or a relational algebra expression if you prefer) that reflects the order of operations that a decent query optimizer would choose.
2. List the join orders (i.e., orders in which pairs of relations can be joined together to compute the query result) that a relational query optimizer will consider. (Assume that the optimizer follows the heuristic of never considering plans that require the computation of cross-products.) Briefly explain how you arrived at your list.
3. Suppose that the following additional information is available: Unclustered B+ tree indexes exist on *Emp.did*, *Emp.sal*, *Dept.floor*, *Dept.did*, and *Finance.did*. The system's statistics indicate that employee salaries range from 10,000 to 60,000, employees enjoy 200 different hobbies, and the company owns two floors in the building. There are a total of 50,000 employees and 5,000 departments (each with corresponding financial information) in the database. The DBMS used by the company has just one join method available, namely, index nested loops.
   (a) For each of the query's base relations (Emp, Dept and Finance) estimate the number of tuples that would be initially selected from that relation if all of the non-join predicates on that relation were applied to it before any join processing begins.
   (b) Given your answer to the preceding question, which of the join orders that are considered by the optimizer has the least estimated cost?

Answer 14.7 The answer to each question is given below.

1.

$$\pi_{D.dname,F.budget}(((\pi_{E.did}(\sigma_{E.sal>=59000,E.hobby="yodelling"}(E)) \\ \bowtie \pi_{D.did,D.dname}(\sigma_{D.floor=1}(D))) \bowtie \pi_{F.budget,F.did}(F))$$

2. There are 2 join orders considered, assuming that the optimizer only consider left-deep joins and ignores cross-products: (D,E,F) and (D,F,E)

3. (a) Emp: card = 50,000, E.sal¿=59,000, E.hobby = "yodelling"
   resulting card = 50000 * 1/50 * 1/200 = 5
   Dept: card = 5000, D.floor = 1
   resulting card = 5000 * 1/2 = 2500
   Finance: card = 5000, there are no non-join predicates
   resulting card = 5000

   (b) Consider the following join methods on the following left-deep tree: $(E \bowtie D) \bowtie F)$.

   The tuples from E will be pipelined, no temporary relations are created.

   First, retrieve the tuples from E with salary ¿= 59,000 using the B-tree index on salary; we estimate 1000 such tuples will be found, with a cost of 1 tree traversal + the cost of retrieving the 1000 tuples (since the index is unclustered) = 3+1000 = 1003. Note, we ignore the cost of scanning the leaves.

   Of these 1000 retrieved tuples, on the fly select only those that have hobby = "yodelling", we estimate there will be 5 such tuples.

   Pipeline these 5 tuples one at a time to D, and using the B-tree index on D.did and the fact the D.did is a key, we can find the matching tuples for the join by searching the Btree and retrieving at most 1 matching tuple, for a total cost of 5(3 + 1) = 20. The resulting cardinality of this join is at most 5.

   Pipeline the estimated 3 tuples of these 5 that have D.floor=1 1 up to F, and use the Btree index on F.did and the fact that F.did is a key to retrieve at most 1 F tuple for each of the 3 pipelined tuples. This costs at most 3(3+1) = 12.

   Ignoring the cost of writing out the final result, we get a total cost of 1003+20+12 = 1035.

Exercise 14.8 Consider the following relational schema and SQL query:

Suppliers(*sid:* `integer`, *sname:* `char(20)`, *city:* `char(20)`)
Supply(*sid:* `integer`, *pid:* `integer`)
Parts(*pid:* `integer`, *pname:* `char(20)`, *price:* `real`)

```
SELECT S.sname, P.pname
FROM   Suppliers S, Parts P, Supply Y
WHERE  S.sid = Y.sid AND Y.pid = P.pid AND
       S.city = 'Madison' AND P.price ≤ 1,000
```

1. What information about these relations will the query optimizer need to select a good query execution plan for the given query?

2. How many different join orders, assuming that cross-products are disallowed, will a System R style query optimizer consider when deciding how to process the given query? List each of these join orders.

3. What indexes might be of help in processing this query? Explain briefly.

4. How does adding `DISTINCT` to the `SELECT` clause affect the plans produced?

5. How does adding `ORDER BY` *sname* to the query affect the plans produced?

6. How does adding `GROUP BY` *sname* to the query affect the plans produced?

Answer 14.8 Answer omitted.

Exercise 14.9 Consider the following scenario:

Emp(*eid:* `integer`, *sal:* `integer`, *age:* `real`, *did:* `integer`)
Dept(*did:* `integer`, *projid:* `integer`, *budget:* `real`, *status:* `char(10)`)
Proj(*projid:* `integer`, *code:* `integer`, *report:* `varchar`)

Assume that each Emp record is 20 bytes long, each Dept record is 40 bytes long, and each Proj record is 2,000 bytes long on average. There are 20,000 tuples in Emp, 5,000 tuples in Dept (note that *did* is not a key), and 1,000 tuples in Proj. Each department, identified by *did*, has 10 projects on average. The file system supports 4,000 byte pages, and 12 buffer pages are available. The following questions are all based on this information. You can assume uniform distribution of values. State any additional assumptions. The cost metric to use is *the number of page I/Os*. Ignore the cost of writing out the final result.

1. Consider the following two queries: "Find all employees with $age = 30$" and "Find all projects with $code = 20$." Assume that the number of qualifying tuples is the same in each case. If you are building indexes on the selected attributes to speed up these queries, for which query is a *clustered* index (in comparison to an *unclustered* index) more important?

2. Consider the following query: "Find all employees with $age > 30$." Assume that there is an unclustered index on *age*. Let the number of qualifying tuples be N. For what values of N is a sequential scan cheaper than using the index?

3. Consider the following query:

```
SELECT *
FROM   Emp E, Dept D
WHERE  E.did=D.did
```

(a) Suppose that there is a clustered hash index on *did* on Emp. List all the plans that are considered and identify the plan with the least estimated cost.

(b) Assume that both relations are sorted on the join column. List all the plans that are considered and show the plan with the least estimated cost.

(c) Suppose that there is a clustered B+ tree index on *did* on Emp and that Dept is sorted on *did*. List all the plans that are considered and identify the plan with the least estimated cost.

4. Consider the following query:

```
SELECT      D.did, COUNT(*)
FROM        Dept D, Proj P
WHERE       D.projid=P.projid
GROUP BY    D.did
```

(a) Suppose that no indexes are available. Show the plan with the least estimated cost.

(b) If there is a hash index on *P.projid* what is the plan with least estimated cost?

(c) If there is a hash index on *D.projid* what is the plan with least estimated cost?

(d) If there is a hash index on *D.projid* and *P.projid* what is the plan with least estimated cost?

(e) Suppose that there is a clustered B+ tree index on *D.did* and a hash index on *P.projid*. Show the plan with the least estimated cost.

(f) Suppose that there is a clustered B+ tree index on *D.did*, a hash index on *D.projid*, and a hash index on *P.projid*. Show the plan with the least estimated cost.

(g) Suppose that there is a clustered B+ tree index on ⟨*D.did, D.projid*⟩ and a hash index on *P.projid*. Show the plan with the least estimated cost.

(h) Suppose that there is a clustered B+ tree index on ⟨*D.projid, D.did*⟩ and a hash index on *P.projid*. Show the plan with the least estimated cost.

5. Consider the following query:

```
SELECT      D.did, COUNT(*)
FROM        Dept D, Proj P
WHERE       D.projid=P.projid AND D.budget>99000
GROUP BY    D.did
```

Assume that department budgets are uniformly distributed in the range 0 to 100,000.

(a) Show the plan with least estimated cost if no indexes are available.

(b) If there is a hash index on *P.projid* show the plan with least estimated cost.

(c) If there is a hash index on *D.budget* show the plan with least estimated cost.

(d) If there is a hash index on *D.projid* and *D.budget* show the plan with least estimated cost.

(e) Suppose that there is a clustered B+ tree index on ⟨*D.did,D.budget*⟩ and a hash index on *P.projid*. Show the plan with the least estimated cost.

(f) Suppose that there is a clustered B+ tree index on *D.did*, a hash index on *D.budget*, and a hash index on *P.projid*. Show the plan with the least estimated cost.

(g) Suppose that there is a clustered B+ tree index on ⟨*D.did, D.budget, D.projid*⟩ and a hash index on *P.projid*. Show the plan with the least estimated cost.

(h) Suppose that there is a clustered B+ tree index on ⟨*D.did, D.projid, D.budget*⟩ and a hash index on *P.projid*. Show the plan with the least estimated cost.

6. Consider the following query:

```
SELECT E.eid, D.did, P.projid
FROM   Emp E, Dept D, Proj P
WHERE  E.sal=50,000 AND D.budget>20,000
       E.did=D.did AND D.projid=P.projid
```

Assume that employee salaries are uniformly distributed in the range 10,009 to 110,008 and that project budgets are uniformly distributed in the range 10,000 to 30,000. There is a clustered index on *sal* for Emp, a clustered index on *did* for Dept, and a clustered index on *projid* for Proj.

(a) List all the one-relation, two-relation, and three-relation subplans considered in optimizing this query.

(b) Show the plan with the least estimated cost for this query.

(c) If the index on Proj were unclustered, would the cost of the preceding plan change substantially? What if the index on Emp or on Dept were unclustered?

Answer 14.9 The reader should calculate actual costs of all alternative plans; in the answers below, we just outline the best plans without detailed cost calculations to prove that these are indeed the best plans.

1. The question specifies that the *number*, rather than the *fraction*, of qualifying tuples is identical for the two queries. Since *Emp* tuples are small, many will fit on a single page; conversely, few (just 2) of the large *Proj* tuples will fit on a page.

Since we wish to minimize the number of page I/Os, it will be an advantage if the *Emp* tuples are clustered with respect to the *age* index (all matching tuples will be retrieved in a few page I/Os). Clustering is not as important for the *Proj* tuples since almost every matching tuple will require a page I/O, even with clustering.

2. The *Emp* relation occupies 100 pages. For an unclustered index retrieving N tuples requires N page I/Os. If more than 100 tuples match, the cost of fetching *Emp* tuples by following pointers in the index data entries exceeds the cost of sequential scan. Using the index also involves about 2 I/Os to get to the right leaf page, and the cost of fetching leaf pages that contain qualifying data entries; this makes scan better than the index with fewer than 100 matches.)

3. (a) One plan is to use (simple or blocked) NL join with E as the outer. Another plan is SM or Hash join. A third plan is to use D as the outer and to use INL; given the clustered hash index on E, this plan will likely be the cheapest.

   (b) The same plans are considered as before, but now, SM join is the best strategy because both relations are sorted on the join column (and all tuples of *Emp* are likely to join with some tuple of *Dept*, and must therefore be fetched at least once, even if INL is used).

   (c) The same plans are considered as before. As in the previous case, SM join is the best: the clustered B+ tree index on *Emp* can be used to efficiently retrieve *Emp* tuples in sorted order.

4. (a) BNL with *Proj* as the outer, followed by sorting on *did* to implement the aggregation. All attributes except *did* can be eliminated during the join but duplicates should not be eliminated!

   (b) Sort *Dept* on *did* first (all other attributes except *projid* can be projected out), then scan while probing *Proj* and counting tuples in each *did* group on-the-fly.

   (c) INL with *Dept* as inner, followed by sorting on *did* to implement the aggregation. Again, all attributes except *did* can be eliminated during the join but duplicates should not be eliminated!

   (d) As in the previous case, INL with *Dept* as inner, followed by sorting on *did* to implement the aggregation. Again, all attributes except *did* can be eliminated during the join but duplicates should not be eliminated!

   (e) Scan *Dept* in *did* order using the clustered B+ tree index while probing *Proj* and counting tuples in each *did* group on-the-fly.

   (f) Same as above.

   (g) Scan the clustered B+ tree index using an index-only scan while probing *Proj* and counting tuples in each *did* group on-the-fly.

   (h) Sort the data entries in the clustered B+ tree index on *Dept*, then scan while probing *Proj* and counting tuples in each *did* group on-the-fly.

5. (a) BNL with *Proj* as the outer with the selection applied on-the-fly, followed by sorting on *did* to implement the aggregation. All attributes except *did* can be eliminated during the join but duplicates should not be eliminated!

   (b) Sort *Dept* on *did* first (while applying the selection and projecting out all other attributes except *projid* in the initial scan), then scan while probing *Proj* and counting tuples in each *did* group on-the-fly.

   (c) Select *Dept* tuples using the index on *budget*, join using INL with *Proj* as inner, projecting out all attributes except *did*. Then sort to implement the aggregation.

   (d) Same as the case with no index; this index does not help.

   (e) Retrieve *Dept* tuples that satisfy the condition on *budget* in *did* order by using the clustered B+ tree index while probing *Proj* and counting tuples in each *did* group on-the-fly.

   (f) Since the condition on *budget* is very selective, even though the index on *budget* is unclustered we retrieve *Dept* tuples using this index, project out the *did* and *projid* fields and sort them by *did*. Then we scan while probing *Proj* and counting tuple sin each *did* gorup on-the-fly.

   (g) Use an index-only scan on the B+ tree and apply the condition on *budget*, while probing *Proj* and counting tuples in each *did* group on-the-fly. Notice that this plan is applicable even if the B+ tree index is *not* clustered. (Within each *did* group, can optimize search for data entries in the index that satisfy the *budget* condition, but this is a minor gain.)

   (h) Use an index-only scan on the B+ tree and apply the condition on *budget*, while probing *Proj* and counting tuples in each *did* group on-the-fly.

6. (a) 1-relation subplans: Clustered index on $E.sal$; Scan *Dept*; and Scan *Proj*.
   2-relation subplans: (i) Clustered index on $E.sal$, probe *Dept* using the index on *did*, apply predicate on $D.budget$ and join. (ii) Scan *Dept*, apply predicate on $D.budget$ and probe *Proj*. (iii) Scan *Proj*, probe *Dept* and apply predicate on $D.budget$ and join.
   3-relation subplans: Join *Emp* and *Dept* and probe *Proj*; Join *Dept* and *Proj* and probe *Emp*.

   (b) The least cost plan is to use the index on $E.sal$ to eliminate most tuples, probe *Dept* using the index on $D.did$, apply the predicate on $D.budget$, probe and join on $Proj.projid$.

   (c) Unclustering the index on *Proj* would increase the number of I/Os but not substantially since the total number of matching *Proj* tuples to be retrieved is small.

15

SCHEMA REFINEMENT AND NORMAL FORMS

Exercise 15.1 Briefly answer the following questions.

1. Define the term *functional dependency.*
2. Give a set of FDs for the relation schema *R(A,B,C,D)* with primary key AB under which R is in 1NF but not in 2NF.
3. Give a set of FDs for the relation schema *R(A,B,C,D)* with primary key AB under which R is in 2NF but not in 3NF.
4. Consider the relation schema *R(A,B,C)*, which has the FD $B \rightarrow C$. If A is a candidate key for R, is it possible for R to be in BCNF? If so, under what conditions? If not, explain why not.
5. Suppose that we have a relation schema *R(A,B,C)* representing a relationship between two entity sets with keys A and B, respectively, and suppose that R has (among others) the FDs $A \rightarrow B$ and $B \rightarrow A$. Explain what such a pair of dependencies means (i.e., what they imply about the relationship that the relation models).

Answer 15.1

1. Let R be a relational schema and let X and Y be two subsets of the set of all attributes of R. We say Y is functionally dependent on X, written X $\rightarrow$ Y, if the Y-values are determined by the X-values. More precisely, for any two tuples r_1 and r_2 in (any instance of) R

$$\pi_X(r_1) = \pi_X(r_2) \quad \Rightarrow \quad \pi_Y(r_1) = \pi_Y(r_2)$$

2. Consider the FD: A $\rightarrow$ C. More generally any (non-trivial) FD: X $\rightarrow \alpha$, with α not equal to A or B, and X = A or X = B will violate 2NF.

3. Consider the FD: D $\rightarrow$ C. More generally any (non-trivial) FD: X $\rightarrow \alpha$, with α not equal to A or B, and X not a proper subset of { A, B } X does not contain AB, will violate 3NF but not 2NF.

4. The only way R could be in BCNF is if B includes a key, *i.e.* B is a key for R.

5. It means that the relationship is one to one. That is, each A entity corresponds to at most one B entity and vice-versa. (In addition, we have the dependency $AB \rightarrow C$, from the semantics of a relationship set.)

Exercise 15.2 Consider a relation R with five attributes $ABCDE$. You are given the following dependencies: $A \rightarrow B$, $BC \rightarrow E$, and $ED \rightarrow A$.

1. List all keys for R.

2. Is R in 3NF?

3. Is R in BCNF?

Answer 15.2 Answer omitted.

Exercise 15.3 Consider the following collection of relations and dependencies. Assume that each relation is obtained through decomposition from a relation with attributes $ABCDEFGHI$ and that all the known dependencies over relation $ABCDEFGHI$ are listed for each question. (The questions are independent of each other, obviously, since the given dependencies over $ABCDEFGHI$ are different.) For each (sub) relation: (a) State the strongest normal form that the relation is in. (b) If it is not in BCNF, decompose it into a collection of BCNF relations.

1. *R1(A,C,B,D,E)*, $A \rightarrow B$, $C \rightarrow D$

2. *R2(A,B,F)*, $AC \rightarrow E$, $B \rightarrow F$

3. *R3(A,D)*, $D \rightarrow G$, $G \rightarrow H$

4. *R4(D,C,H,G)*, $A \rightarrow I$, $I \rightarrow A$

5. *R5(A,I,C,E)*

Answer 15.3

1. Not in 3NF. BCNF decomposition: AB, CD, ACE.

2. Not in 3NF. BCNF decomposition: AB, BF

3. BCNF.

4. BCNF.

5. BCNF.

Exercise 15.4 Suppose that we have the following three tuples in a legal instance of a relation schema S with three attributes ABC (listed in order): (1,2,3), (4,2,3), and (5,3,3).

1. Which of the following dependencies can you infer does *not* hold over schema S?

   (a) $A \rightarrow B$ (b) $BC \rightarrow A$ (c) $B \rightarrow C$

2. Can you identify any dependencies that hold over S?

Answer 15.4 Answer omitted.

Exercise 15.5 Suppose you are given a relation R with four attributes, $ABCD$. For each of the following sets of FDs, assuming those are the only dependencies that hold for R, do the following: (a) Identify the candidate key(s) for R. (b) Identify the best normal form that R satisfies (1NF, 2NF, 3NF, or BCNF). (c) If R is not in BCNF, decompose it into a set of BCNF relations that preserve the dependencies.

1. $C \rightarrow D$, $C \rightarrow A$, $B \rightarrow C$
2. $B \rightarrow C$, $D \rightarrow A$
3. $ABC \rightarrow D$, $D \rightarrow A$
4. $A \rightarrow B$, $BC \rightarrow D$, $A \rightarrow C$
5. $AB \rightarrow C$, $AB \rightarrow D$, $C \rightarrow A$, $D \rightarrow B$

Answer 15.5

1. (a) Candidate keys: B
   (b) R is in 2NF but not 3NF.
   (c) $C \rightarrow D$ and $C \rightarrow A$ both cause violations of BCNF. One way to obtain a (lossless) join preserving decomposition is to decompose R into AC, BC, and CD.

2. (a) Candidate keys: BD
   (b) R is in 1NF but not 2NF.
   (c) Both $B \rightarrow C$ and $D \rightarrow A$ cause BCNF violations. The decomposition: AD, BC, BD (obtained by first decomposing to AD, BCD) is BCNF and lossless and join-preserving.

3. (a) Candidate keys: ABC, BCD
   (b) R is in 3NF but not BCNF.
   (c) ABCD is not in BCNF since $D \to A$ and D is not a key. However if we split up R as AD, BCD we cannot preserve the dependency $ABC \to D$. So there is no BCNF decomposition.
4. (a) Candidate keys: A
   (b) R is in 2NF but not 3NF (because of the FD: BC $\to$ D).
   (c) $BC \to D$ violates BCNF since BC does not contain a key. So we split up R as in: BCD, ABC.
5. (a) Candidate keys: AB, BC, CD, AD
   (b) R is in 3NF but not BCNF (because of the FD: C $\to$ A).
   (c) $C \to A$ and $D \to B$ both cause violations. So decompose into: AC, BCD but this does not preserve $AB \to C$ and $AB \to D$, and BCD is still not BCNF because $D \to B$. So we need to decompose further into: AC, BD, CD. Now add ABC and ABD to get the final decomposition: AC, BD, CD, ABC, ABD which is BCNF, lossless and join-preserving.

Exercise 15.6 Consider the attribute set $R = ABCDEGH$ and the FD set $F = \{AB \to C, AC \to B, AD \to E, B \to D, BC \to A, E \to G\}$.

1. For each of the following attribute sets, do the following: (i) Compute the set of dependencies that hold over the set and write down a minimal cover. (ii) Name the strongest normal form that is not violated by the relation containing these attributes. (iii) Decompose it into a collection of BCNF relations if it is not in BCNF.

   (a) ABC (b) $ABCD$ (c) $ABCEG$ (d) $DCEGH$ (e) $ACEH$

2. Which of the following decompositions of $R = ABCDEG$, with the same set of dependencies F, is (a) dependency-preserving? (b) lossless-join?
   (a) $\{AB, BC, ABDE, EG\}$
   (b) $\{ABC, ACDE, ADG\}$

Answer 15.6 Answer omitted.

Exercise 15.7 Let R be decomposed into R_1, R_2, ..., R_n. Let F be a set of FDs on R.

1. Define what it means for F to *be preserved* in the set of decomposed relations.

2. Describe a polynomial-time algorithm to test dependency-preservation.

3. Projecting the FDs stated over a set of attributes X onto a subset of attributes Y requires that we consider the closure of the FDs. Give an example where considering the closure is important in testing dependency-preservation; that is, considering just the given FDs gives incorrect results.

Answer 15.7

1. Let F_i denote the projection of F on R_i. F is *preserved* if the closure of the (union of) the F_i's equals F (note that F is always a superset of this closure.)

2. We shall describe an algorithm for testing dependency preservation which is polynomial in the cardinality of F. For each dependency X $\rightarrow$Y $\in$ F check if it is in F as follows: start with the set S (of attributes in) X. For each relation R_i, compute the closure of $S \cap R_i$ relative to F and project this closure to the attributes of R_i. If this results in additional attributes, add them to S. Do this repeatedly until there is no change to S.

3. There is an example in the text.

Exercise 15.8 Consider a relation R that has three attributes ABC. It is decomposed into relations R_1 with attributes AB and R_2 with attributes BC.

1. State the definition of a lossless-join decomposition with respect to this example. Answer this question concisely by writing a relational algebra equation involving R, R_1, and R_2.

2. Suppose that $B \rightarrow\rightarrow C$. Is the decomposition of R into R_1 and R_2 lossless-join? Reconcile your answer with the observation that neither of the FDs $R_1 \cap R_2 \rightarrow R_1$ nor $R_1 \cap R_2 \rightarrow R_2$ hold, in light of the simple test offering a necessary and sufficient condition for lossless-join decomposition into two relations in Section 15.6.1.

3. If you are given the following instances of R_1 and R_2, what can you say about the instance of R from which these were obtained? Answer this question by listing tuples that are definitely in R and listing tuples that are possibly in R.

   Instance of R_1 = {(5,1), (6,1)}
   Instance of R_2 = {(1,8), (1,9)}

   Can you say that attribute B definitely *is* or *is not* a key for R?

Answer 15.8 Answer omitted.

Exercise 15.9 Suppose you are given a relation *R(A,B,C,D)*. For each of the following sets of FDs, assuming they are the only dependencies that hold for R, do the following: (a) Identify the candidate key(s) for R. (b) State whether or not the proposed decomposition of R into smaller relations is a good decomposition, and briefly explain why or why not.

1. $B \rightarrow C$, $D \rightarrow A$; decompose into BC and AD.
2. $AB \rightarrow C$, $C \rightarrow A$, $C \rightarrow D$; decompose into ACD and BC.
3. $A \rightarrow BC$, $C \rightarrow AD$; decompose into ABC and AD.
4. $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$; decompose into AB and ACD.
5. $A \rightarrow B$, $B \rightarrow C$, $C \rightarrow D$; decompose into AB, AD and CD.

Answer 15.9

1. Candidate key(s): BD. The decomposition into BC and AD is unsatisfactory because it is lossy (the join of BC and AD is the cartesian product which could be much bigger than ABCD)
2. Candidate key(s): AB, BC. The decomposition into ACD and BC is lossless since ACD $\cap$ BC (which is C) $\rightarrow$ACD. The projection of the FD's on ACD include C $\rightarrow$D, C $\rightarrow$A (so C is a key for ACD) and the projection of FD on BC produces no nontrivial dependencies. In particular this is a BCNF decomposition (check that R is not!). However, it is not dependency preserving since the dependency AB $\rightarrow$C is not preserved. So to enforce preservation of this dependency (if we do not want to use a join) we need to add ABC which introduces redundancy. So implicitly there is some redundancy across relations (although none inside ACD and BC).
3. Candidate key(s): A, C. Since A and C are both candidate keys for R, it is already in BCNF. So from a normalization standpoint it makes no sense to decompose R further.
4. Candidate key(s): A. The projection of the dependencies on AB are: A $\rightarrow$B and those on ACD are: A $\rightarrow$C and C $\rightarrow$D (rest follow from these). The scheme ACD is not even in 3NF, since C is not a superkey, and D is not part of a key. This is a lossless-join decomposition (since A is a key), but not dependency preserving, since B $\rightarrow$C is not preserved.
5. Candidate key(s): A (just as before) This is a lossless BCNF decomposition (easy to check!) This is, however, not dependency preserving (B consider $\rightarrow$C). So it is not free of (implied) redundancy. This is not the best decomposition (the decomposition AB, BC, CD is better.)

Exercise 15.10 Suppose that we have the following four tuples in a relation S with three attributes ABC: (1,2,3), (4,2,3), (5,3,3), (5,3,4). Which of the following functional ($\rightarrow$) and multivalued ($\rightarrow\rightarrow$) dependencies can you infer does *not* hold over relation S?

1. $A \rightarrow B$

2. $A \rightarrow\rightarrow B$
3. $BC \rightarrow A$
4. $BC \rightarrow\rightarrow A$
5. $B \rightarrow C$
6. $B \rightarrow\rightarrow C$

Answer 15.10 Answer omitted.

Exercise 15.11 Consider a relation R with five attributes $ABCDE$.

1. For each of the following instances of R, state whether (a) it violates the FD $BC \rightarrow D$, and (b) it violates the MVD $BC \rightarrow\rightarrow D$:
   (a) { } (i.e., empty relation)
   (b) {(a,2,3,4,5), (2,a,3,5,5)}
   (c) {(a,2,3,4,5), (2,a,3,5,5), (a,2,3,4,6)}
   (d) {(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5)}
   (e) {(a,2,3,4,5), (2,a,3,7,5), (a,2,3,4,6)}
   (f) {(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5), (a,2,3,6,6)}
   (g) {(a,2,3,4,5), (a,2,3,6,5), (a,2,3,6,6), (a,2,3,4,6)}
2. If each instance for R listed above is legal, what can you say about the FD $A \rightarrow B$?

Answer 15.11

1. **Note**: The answer sometimes depends on the value of a. Unless otherwise mentioned, the answer applies to all values of a.

   (a) { } (i.e., empty relation):
   does not violate either dependency.
   (b) {(a,2,3,4,5), (2,a,3,5,5)}:
   BC $\rightarrow$D is violated if and only if a = 2.
   BC $\rightarrow\rightarrow$D is not violated (for any value of a)
   (c) {(a,2,3,4,5), (2,a,3,5,5), (a,2,3,4,6)}:
   BC $\rightarrow$D is violated if a = 2 (otherwise not).
   If a = 2 then BC $\rightarrow\rightarrow$D is violated (consider the tuples (2,a,3,5,5) and (a,2,3,4,6); if a equals 2 must also have (2,a,3,5,6))

(d) {(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5)}:
BC →D is violated (consider the first and the third tuples ((a,2,3,4,5) and (a,2,3,6,5)).
BC →→D is not violated.

(e) {(a,2,3,4,5), (2,a,3,7,5), (a,2,3,4,6)}:
If a = 2 then BC →D is violated (otherwise it is not).
If a = 2 then BC →→D is violated (otherwise it is not). To prove this look at the last two tuples; there must also be a tuple (2,a,3,7,6) for BC →→to hold.

(f) {(a,2,3,4,5), (2,a,3,4,5), (a,2,3,6,5), (a,2,3,6,6)}:
BC →D does not hold. (Consider the first and the third tuple).
BC →→C is violated. Consider the 1st and the 4th tuple. For this dependency to hold there should be a tuple (a,2,3,4,6).

(g) {(a,2,3,4,5), (a,2,3,6,5), (a,2,3,6,6), (a,2,3,4,6)}:
BC →D does not hold. (Consider the first and the third tuple).
BC →→C is not violated.

2. We can*not* say anything about the functional dependency A →B.

Exercise 15.12 JDs are motivated by the fact that sometimes a relation that cannot be decomposed into two smaller relations in a lossless-join manner can be so decomposed into three or more relations. An example is a relation with attributes *supplier*, *part*, and *project*, denoted SPJ, with no FDs or MVDs. The JD $\bowtie \{SP,\ PJ,\ JS\}$ holds.

From the JD, the set of relation schemes SP, PJ, and JS is a lossless-join decomposition of SPJ. Construct an instance of SPJ to illustrate that no two of these schemes suffice.

Answer 15.12 Answer omitted.

Exercise 15.13 Consider a relation R with attributes $ABCDE$. Let the following FDs be given: $A \rightarrow BC$, $BC \rightarrow E$, and $E \rightarrow DA$. Similarly, let S be a relation with attributes $ABCDE$ and let the following FDs be given: $A \rightarrow BC$, $B \rightarrow E$, and $E \rightarrow DA$. (Only the second dependency differs from those that hold over R.) You do not know whether or which other (join) dependencies hold.

1. Is R in BCNF?

2. Is R in 4NF?

3. Is R in 5NF?

4. Is S in BCNF?

5. Is S in 4NF?
6. Is S in 5NF?

Answer 15.13

1. The schema R has keys A, E and BC. It follows that R is indeed in BCNF.
2. By Exercise 23, Part 1 it follows that R is also in 4NF (since the relation scheme has a single-attribute key).
3. R is in 5NF because the schema does not have any JD (besides those that are implied by the FD's of the schema; but these cannot violate the 5NF condition). Note that this alternative argument may be used in some of the other parts of this problem as well.
4. The schema S has keys A, B and E. It follows that S is indeed in BCNF.
5. By exercise 23 (part 1) it follows that S is also in 4NF (since the relation scheme has a single-attribute key).
6. By exercise 23 (part 2) it follows that S is also in 5NF (since each key is a single-attribute key.)

Exercise 15.14 Let us say that an FD $X \rightarrow Y$ is *simple* if Y is a single attribute.

1. Replace the FD $AB \rightarrow CD$ by the smallest equivalent collection of simple FDs.
2. Prove that every FD $X \rightarrow Y$ in a set of FDs F can be replaced by a set of simple FDs such that F^+ is equal to the closure of the new set of FDs.

Answer 15.14 Answer omitted.

Exercise 15.15 Prove that Armstrong's Axioms are sound and complete for FD inference. That is, show that repeated application of these axioms on a set F of FDs produces exactly the dependencies in F^+.

Answer 15.15 Proof omitted.

Exercise 15.16 Describe a linear-time (in the size of the set of FDs, where the size of each FD is the number of attributes involved) algorithm for finding the attribute closure of a set of attributes with respect to a set of FDs.

Answer 15.16 Answer omitted.

Exercise 15.17 Consider a scheme R with FDs F that is decomposed into schemes with attributes X and Y. Show that this is dependency-preserving if $F \subseteq (F_X \cup F_Y)^+$.

Answer 15.17 We need to show that $F^+ = (F_X \cup F_Y)^+$. Both containments are based on two observations:

1. If $A \subseteq B$ are two sets of FD's then $A^+ \subseteq B^+$ and
2. $A^{++} = A^+$.

The includsion $(F_X \cup F_Y)^+ \subseteq F^+$ follows from observing that, by definition, $F_X \subseteq F^+$ and $F_Y \subseteq F^+$ so that $F_X \cup F_Y \subseteq F^+$ (now apply observations 1 and 2).

The other containment, $F^+ \subseteq (F_X \cup F_Y)^+$ follows from the hypothesis, $F \subseteq (F_X \cup F_Y)^+$ and observations 1 and 2.

Exercise 15.18 Let R be a relation schema with a set F of FDs. Prove that the decomposition of R into R_1 and R_2 is lossless-join if and only if F^+ contains $R_1 \cap R_2 \rightarrow R_1$ or $R_1 \cap R_2 \rightarrow R_2$.

Answer 15.18 Answer omitted.

Exercise 15.19 Prove that the optimization of the algorithm for lossless-join, dependency-preserving decomposition into 3NF relations (Section 15.7.2) is correct.

Answer 15.19

Exercise 15.20 Prove that the 3NF synthesis algorithm produces a lossless-join decomposition of the relation containing all the original attributes.

Answer 15.20 Answer omitted.

Exercise 15.21 Prove that an MVD $X \rightarrow\rightarrow Y$ over a relation R can be expressed as the join dependency $\bowtie \{XY,\ X(R-Y)\}$.

Answer 15.21 Write $Z = R - Y$. Thus, $R = YXZ$. $X \rightarrow\rightarrow Y$ says that if $(y_1, x, z_1), (y_2, x, z_2) \in R$ then $(y_1, x, z_2), (y_2, x, z_1)$ also $\in R$. But this is precisely the same as saying $R = \bowtie \{\ XY,\ X(R-Y)\ \}$.

Exercise 15.22 Prove that if R has only one key, it is in BCNF if and only if it is in 3NF.

Answer 15.22 Answer omitted.

Exercise 15.23 Prove that if R is in 3NF and every key is simple, then R is in BCNF.
Answer 15.23 Since every key is simple, then we know that for any FD that satisfies $X \rightarrow A$, where A is part of some key implies that A is a key. By the definition of an FD, if X is known, then A is known. This means that if X is known, we know a key for the relation, so X must be a superkey. This satisfies all of the properties of BCNF.

Exercise 15.24 Prove these statements:

1. If a relation scheme is in BCNF and at least one of its keys consists of a single attribute, it is also in 4NF.
2. If a relation scheme is in 3NF and each key has a single attribute, it is also in 5NF.

Answer 15.24 Answer omitted.

Exercise 15.25 Give an algorithm for testing whether a relation scheme is in BCNF. The algorithm should be polynomial in the size of the set of given FDs. (The *size* is the sum over all FDs of the number of attributes that appear in the FD.) Is there a polynomial algorithm for testing whether a relation scheme is in 3NF?
Answer 15.25 Fix some instance of the schema r, and take two tuples λ and μ with

$$\lambda[X] \;=\; \mu[X]$$

Let U denote the set of all the attributes of the schema and let

$$z_1 = \lambda[Z] \quad \text{and} \quad z_2 = \mu[Z].$$

We need to show that $z_1 = z_2$.

The MVD $X \rightarrow\rightarrow Z$ implies the existence of a tuple ν such that

$$\nu[X] = \lambda[X] = \mu[X],\ \nu[Z] = \lambda[Z] = z_1 \text{ and } \nu[U - XZ] = \mu[U - XZ] = z_2.$$

Since, Y and Z are disjoint, we have

$$Y = (Y \cap X) \cup (Y \cap (U - XZ)).$$

Also, ν and μ agree on X as well as on $U - XZ$. Thus, we conclude: $\nu[Y] = \mu[Y]$. From the FD $Y \rightarrow Z$ we can then conclude that $\nu[Z] \;=\; \mu[Z]$, or $z_1 \;=\; z_2$.

Exercise 15.26 Give an algorithm for testing whether a relation scheme is in BCNF. The algorithm should be polynomial in the size of the set of given FDs. (The ‘size’ is the sum over all FDs of the number of attributes that appear in the FD.) Is there a polynomial algorithm for testing whether a relation scheme is in 3NF?
Answer 15.26 Answer omitted.

16

PHYSICAL DATABASE DESIGN AND TUNING

Exercise 16.1 Consider the following relations:

Emp(*eid:* `integer`, *ename:* `varchar`, *sal:* `integer`, *age:* `integer`, *did:* `integer`)
Dept(*did:* `integer`, *budget:* `integer`, *floor:* `integer`, *mgr_eid:* `integer`)

Salaries range from $10,000 to $100,000, ages vary from 20 to 80, each department has about five employees on average, there are 10 floors, and budgets vary from $10,000 to $1,000,000. You can assume uniform distributions of values.

For each of the following queries, which of the listed index choices would you choose to speed up the query? If your database system does not consider index-only plans (i.e., data records are always retrieved even if enough information is available in the index entry), how would your answer change? Explain briefly.

1. Query: *Print ename, age, and sal for all employees.*
   (a) Clustered, dense hash index on $\langle ename, age, sal\rangle$ fields of Emp.
   (b) Unclustered hash index on $\langle ename, age, sal\rangle$ fields of Emp.
   (c) Clustered, sparse B+ tree index on $\langle ename, age, sal\rangle$ fields of Emp.
   (d) Unclustered hash index on $\langle eid, did\rangle$ fields of Emp.
   (e) No index.
2. Query: *Find the dids of departments that are on the 10th floor and that have a budget of less than $15,000.*
   (a) Clustered, dense hash index on the $floor$ field of Dept.
   (b) Unclustered hash index on the $floor$ field of Dept.
   (c) Clustered, dense B+ tree index on $\langle floor, budget\rangle$ fields of Dept.
   (d) Clustered, sparse B+ tree index on the *budget* field of Dept.

(e) No index.

3. Query: *Find the names of employees who manage some department and have a salary greater than \$12,000.*

   (a) Clustered, sparse B+ tree index on the *sal* field of Emp.

   (b) Clustered hash index on the *did* field of Dept.

   (c) Unclustered hash index on the *did* field of Dept.

   (d) Unclustered hash index on the *did* field of Emp.

   (e) Clustered B+ tree index on *sal* field of Emp and clustered hash index on the *did* field of Dept.

4. Query: *Print the average salary for each department.*

   (a) Clustered, sparse B+ tree index on the *did* field of Emp.

   (b) Clustered, dense B+ tree index on the *did* field of Emp.

   (c) Clustered, dense B+ tree index on $\langle did, sal\rangle$ fields of Emp.

   (d) Unclustered hash index on $\langle did, sal\rangle$ fields of Emp.

   (e) Clustered, dense B+ tree index on the *did* field of Dept.

Answer 16.1 The answer to each question is given below.

1. We should create an unclustered hash index on $\langle ename, age, sal\rangle$ fields of Emp (b) since then we could do an index only scan. If our system does not include index only plans then we shouldn't create an index for this query (e). Since this query requires us to access all the Emp records, an index won't help us any, and so should we access the records using a filescan.

2. We should create a clustered dense B+ tree index (c) on $\langle floor, budget\rangle$ fields of Dept, since the records would be ordered on these fields then. So when executing this query, the first record with $floor = 10$ must be retrieved, and then the other records with $floor = 10$ can be read in order of budget. Note that this plan, which is the best for this query, is not an index-only plan (must look up dids).

3. We should create a hash index on the *eid* field of Emp (d) since then we can do a filescan on Dept and hash each manager id into Emp and check to see if the salary is greater than 12,000 dollars. None of the indexes offered lend themselves to index-only scans, so it doesn't matter if they are allowed or not.

4. For this query we should create a dense clustered B+ tree (c) index on $\langle did, sal\rangle$ fields of the Emp relation, so we can do an index only scan. (An unclustered index would be sufficient, but is not included in the list of index choices for this question.) If index-only scans are not allowed, we should then create a clustered sparse B+ tree index on the *did* field of Emp (a). This index will be just as

efficient as an index on $\langle did, sal\rangle$ since both will involve accessing the data records in *did* order. However, since we now have only one attribute in the search key, it will be updated less often.

Exercise 16.2 Consider the following relation:

Emp(<u>*eid:* `integer`</u>, *sal:* `integer`, *age:* `real`, *did:* `integer`)

There is a clustered index on *eid* and an unclustered index on *age*.

1. Which factors would you consider in deciding whether to make an index on a relation a clustered index? Would you always create at least one clustered index on every relation?
2. How would you use the indexes to enforce the constraint that *eid* is a key?
3. Give an example of an update that is *definitely speeded up* because of the available indexes. (English description is sufficient.)
4. Give an example of an update that is *definitely slowed down* because of the indexes. (English description is sufficient.)
5. Can you give an example of an update that is neither speeded up nor slowed down by the indexes?

Answer 16.2 Answer omitted.

Exercise 16.3 Consider the following BCNF schema for a portion of a simple corporate database (type information is not relevant to this question and is omitted):

Emp (<u>*eid*</u>, *ename, addr, sal, age, yrs, deptid*)
Dept (<u>*did*</u>, *dname, floor, budget*)

Suppose you know that the following queries are the six most common queries in the workload for this corporation and that all six are roughly equivalent in frequency and importance:

- List the id, name, and address of employees in a user-specified age range.
- List the id, name, and address of employees who work in the department with a user-specified department name.
- List the id and address of employees with a user-specified employee name.

- List the overall average salary for employees.
- List the average salary for employees of each age; that is, for each age in the database, list the age and the corresponding average salary.
- List all the department information, ordered by department floor numbers.

1. Given this information, and assuming that these queries are more important than any updates, design a physical schema for the corporate database that will give good performance for the expected workload. In particular, decide which attributes will be indexed and whether each index will be a clustered index or an unclustered index. Assume that B+ tree indexes are the only index type supported by the DBMS and that both single- and multiple-attribute keys are permitted. Specify your physical design by identifying the attributes that you recommend indexing on via clustered or unclustered B+ trees.
2. Redesign the physical schema assuming that the set of important queries is changed to be the following:
   - List the id and address of employees with a user-specified employee name.
   - List the overall maximum salary for employees.
   - List the average salary for employees by department; that is, for each *deptid* value, list the *deptid* value and the average salary of employees in that department.
   - List the sum of the budgets of all departments by floor; that is, for each floor, list the floor and the sum.

Answer 16.3 The answer to each question is given below.

1. - If we create a dense unclustered B+ tree index on $\langle age, sal\rangle$ of the Emp relation we will be able to do an index-only scan to answer the 5th query. A hash index would not serve our purpose here, since the data entries will not be ordered by *age*! If index only scans are not allowed create a clustered B+ tree index on just the *age* field of Emp.
   - We should create an unclustered B+Tree index on *deptid* of the Emp relation and another unclustered index on $\langle dname, did\rangle$ in the Dept relation. Then, we can do an index only search on Dept and then get the Emp records with the proper *deptid*'s for the second query.
   - We should create an unclustered index on *ename* of the Emp relation for the third query.
   - We want a clustered sparse B+ tree index on $floor$ of the Dept index so we can get the department on each floor in $floor$ order for the sixth query.

- Finally, a dense unclustered index on *sal* will allow us to average the salaries of all employees using an index only-scan. However, the dense unclustered B+ tree index on $\langle age, sal\rangle$ that we created to support Query (5) can also be used to compute the average salary of all employees, and is almost as good for this query as an index on just *sal*. So we should not create a separate index on just *sal*.

2. - We should create an unclustered B+Tree index on *ename* for the Emp relation so we can efficiently find employees with a particular name for the first query. This is not an index-only plan.
   - An unclustered B+ tree index on *sal* for the Emp relation will help find the maximum salary for the second query. (This is better than a hash index because the aggregate operation involved is `MAX`—we can simply go down to the rightmost leaf page in the B+ tree index.) This is not an index-only plan.
   - We should create a dense unclustered B+ tree index on $\langle deptid, sal\rangle$ of the Emp relation so we can do an index-only scan on all of a department's employees. If index only plans are not supported, a sparse, clustered B+ tree index on *deptid* would be best. It would allow us to retrieve tuples by *deptid*.
   - We should create a dense, unclustered index on $\langle floor, budget\rangle$ for Dept. This would allow us to sum budgets by floor using an index only plan. If index-only plans are not supported, we should create a sparse clustered B+ tree index on *floor* for the Dept relation, so we can find the departments on each floor in order by floor.

Exercise 16.4 Consider the following BCNF relational schema for a portion of a university database (type information is not relevant to this question and is omitted):

Prof(*<u>ssno</u>, pname, office, age, sex, specialty, dept_did*)
Dept(*<u>did</u>, dname, budget, num_majors, chair_ssno*)

Suppose you know that the following queries are the five most common queries in the workload for this university and that all five are roughly equivalent in frequency and importance:

- List the names, ages, and offices of professors of a user-specified sex (male or female) who have a user-specified research specialty (e.g., *recursive query processing*). Assume that the university has a diverse set of faculty members, making it very uncommon for more than a few professors to have the same research specialty.
- List all the department information for departments with professors in a user-specified age range.

- List the department id, department name, and chairperson name for departments with a user-specified number of majors.
- List the lowest budget for a department in the university.
- List all the information about professors who are department chairpersons.

These queries occur much more frequently than updates, so you should build whatever indexes you need to speed up these queries. However, you should not build any unnecessary indexes, as updates will occur (and would be slowed down by unnecessary indexes). Given this information, design a physical schema for the university database that will give good performance for the expected workload. In particular, decide which attributes should be indexed and whether each index should be a clustered index or an unclustered index. Assume that both B+ trees and hashed indexes are supported by the DBMS and that both single- and multiple-attribute index search keys are permitted.

1. Specify your physical design by identifying the attributes that you recommend indexing on, indicating whether each index should be clustered or unclustered and whether it should be a B+ tree or a hashed index.
2. Redesign the physical schema assuming that the set of important queries is changed to be the following:
   - List the number of different specialties covered by professors in each department, by department.
   - Find the department with the fewest majors.
   - Find the youngest professor who is a department chairperson.

Answer 16.4 Answer omitted.

Exercise 16.5 Consider the following BCNF relational schema for a portion of a company database (type information is not relevant to this question and is omitted):

Project(*pno*, *proj_name*, *proj_base_dept*, *proj_mgr*, *topic*, *budget*)
Manager(*mid*, *mgr_name*, *mgr_dept*, *salary*, *age*, *sex*)

Note that each project is based in some department, each manager is employed in some department, and the manager of a project need not be employed in the same department (in which the project is based). Suppose you know that the following queries are the five most common queries in the workload for this university and that all five are roughly equivalent in frequency and importance:

- List the names, ages, and salaries of managers of a user-specified sex (male or female) working in a given department. You can assume that while there are many departments, each department contains very few project managers.

- List the names of all projects with managers whose ages are in a user-specified range (e.g., younger than 30).
- List the names of all departments such that a manager in this department manages a project based in this department.
- List the name of the project with the lowest budget.
- List the names of all managers in the same department as a given project.

These queries occur much more frequently than updates, so you should build whatever indexes you need to speed up these queries. However, you should not build any unnecessary indexes, as updates will occur (and would be slowed down by unnecessary indexes). Given this information, design a physical schema for the company database that will give good performance for the expected workload. In particular, decide which attributes should be indexed and whether each index should be a clustered index or an unclustered index. Assume that both B+ trees and hashed indexes are supported by the DBMS, and that both single- and multiple-attribute index keys are permitted.

1. Specify your physical design by identifying the attributes that you recommend indexing on, indicating whether each index should be clustered or unclustered and whether it should be a B+ tree or a hashed index.
2. Redesign the physical schema assuming that the set of important queries is changed to be the following:
   - Find the total of the budgets for projects managed by each manager; that is, list *proj_mgr* and the total of the budgets of projects managed by that manager, for all values of *proj_mgr*.
   - Find the total of the budgets for projects managed by each manager but only for managers who are in a user-specified age range.
   - Find the number of male managers.
   - Find the average age of managers.

Answer 16.5 The answer to each question is given below.

1. - For the first query, we should create a dense unclustered hash index on *mgr_dept* for the Manager relation. We omit *sex* from the key in this index since it is not very selective; however, including it is probably not very expensive since this field is unlikely to be updated.
   - We should create a unclustered B+ tree index on $\langle age, mgr\_dept, mid\rangle$ for the Manager relation, and an unclustered hash index on $\langle proj\_base\_dept, proj\_mgr\rangle$ for the Project relation. We can do an index only scan to find managers whose

age is in the specified range, and then hash into the Project relation to get the project names. If index only scans are not supported, the index on manager should be a clustered index on *age*.

- For the third query we don't need a new index. We can scan all managers and use the hash index on $\langle proj\_base\_dept, proj\_mgr \rangle$ on the Project relation to check if $mgr\_dept = proj\_base\_dept$.
- We can create an unclustered B+ tree index on *budget* in the Project relation and then go down the tree to find the lowest budget for the fourth query.
- For the fifth query, we should create dense unclustered hash index on *pno* for the Project relation. We can can get the *proj_base_dept* of the project by using this index, and then use the hash index on *mgr_dept* to get the managers in this department. Note that an index on $\langle pno, proj\_base\_dept$ for Project would allow us to do an index only scan on Project. However, since there is exactly one base department for each project (*pno* is the key) this is not likely to be significantly faster. (It does save us one I/O per project.)

2.
- For the first query, we should create an unclustered B+Tree index on $\langle proj\_mgr, budget \rangle$ for the Project relation. An index only scan can then be used to solve the query. If index only scans are not supported, a clustered index on *proj_mgr* would be best.
- If we create a sparse clustered B+ tree index on $\langle age, mid \rangle$ for Manager, we can do an index only scan on this index to find the ids of managers in the given range. Then, we can use an index only scan of the B+Tree index on $\langle proj\_mgr, budget \rangle$ to compute the total of the budgets of the projects that each of these managers manages. If index only scans are not supported, the index on Manager should be a clustered B+ tree index on *age*.
- An unclustered hash index on *sex* will divide the managers by sex and allow us to count the number that are male using an index only scan. If index only scans are not allowed, then no index will help us for the third query.
- We should create an unclustered hash index on *age* for the fourth query. All we need to do is average the ages using an index-only scan. If index-only plans are not allowed no index will help us.

Exercise 16.6 The Globetrotters Club is organized into chapters. The president of a chapter can never serve as the president of any other chapter, and each chapter gives its president some salary. Chapters keep moving to new locations, and a new president is elected when (and only when) a chapter moves. The above data is stored in a relation G(C,S,L,P), where the attributes are chapters (C), salaries (S), locations (L), and presidents (P). Queries of the following form are frequently asked, and you *must* be able to answer them without computing a join: "Who was the president of chapter X when it was in location Y?"

1. List the FDs that are given to hold over G.
2. What are the candidate keys for relation G?
3. What normal form is the schema G in?
4. Design a good database schema for the club. (Remember that your design *must* satisfy the query requirement stated above!)
5. What normal form is your good schema in? Give an example of a query that is likely to run slower on this schema than on the relation G.
6. Is there a lossless-join, dependency-preserving decomposition of G into BCNF?
7. Is there ever a good reason to accept something less than 3NF when designing a schema for a relational database? Use this example, if necessary adding further constraints, to illustrate your answer.

Answer 16.6 Answer omitted.

Exercise 16.7 Consider the following BCNF relation, which lists the ids, types (e.g., nuts or bolts), and costs of various parts, along with the number that are available or in stock:

Parts (*<u>pid</u>, pname, cost, num_avail*)

You are told that the following two queries are extremely important:

- Find the total number available by part type, for all types. (That is, the sum of the *num_avail* value of all nuts, the sum of the *num_avail* value of all bolts, etc.)
- List the *pid*s of parts with the highest cost.

1. Describe the physical design that you would choose for this relation. That is, what kind of a file structure would you choose for the set of Parts records, and what indexes would you create?
2. Suppose that your customers subsequently complain that performance is still not satisfactory (given the indexes and file organization that you chose for the Parts relation in response to the previous question). Since you cannot afford to buy new hardware or software, you have to consider a schema redesign. Explain how you would try to obtain better performance by describing the schema for the relation(s) that you would use and your choice of file organizations and indexes on these relations.
3. How would your answers to the above two questions change, if at all, if your system did not support indexes with multiple-attribute search keys?

Answer 16.7 The answer to each question is given below.

1. A heap file structure could be used for the relation Parts. A dense unclustered B+Tree index on $\langle pname, num\_avail \rangle$ and a dense unclustered B+ Tree index on $\langle cost, pid \rangle$ can be created to efficiently answers the queries.

2. The problem could be that the optimizer may not be considering the index only plans that could be obtained using the previously described schema. So we can instead create clustered indexes on $\langle pid, cost \rangle$ and $\langle pname, num\_avail \rangle$. To do this we have to vertically partition the relation into two relations like Parts1(pid, cost) and Parts2(pid, pname, num_avail). (If the indexes themselves have not been implemented properly, then we can instead use sorted file organizations for these two split relations).

3. If the multi attribute keys are not allowed then we can have a clustered B+ Tree indexes on *cost* and on *pname* on the two relations.

Exercise 16.8 Consider the following BCNF relations, which describe employees and departments that they work in:

Emp (*eid*, *sal*, *did*)
Dept (*did*, *location*, *budget*)

You are told that the following queries are extremely important:

- Find the location where a user-specified employee works.
- Check whether the budget of a department is greater than the salary of each employee in that department.

1. Describe the physical design that you would choose for this relation. That is, what kind of a file structure would you choose for these relations, and what indexes would you create?

2. Suppose that your customers subsequently complain that performance is still not satisfactory (given the indexes and file organization that you chose for the relations in response to the previous question). Since you cannot afford to buy new hardware or software, you have to consider a schema redesign. Explain how you would try to obtain better performance by describing the schema for the relation(s) that you would use and your choice of file organizations and indexes on these relations.

3. Suppose that your database system has very inefficient implementations of index structures. What kind of a design would you try in this case?

Answer 16.8 Answer omitted.

Exercise 16.9 Consider the following BCNF relations, which describe departments in a company and employees:

Dept(*did*, *dname, location, managerid*)
Emp(*eid*, *sal*)

You are told that the following queries are extremely important:

- List the names and ids of managers for each department in a user-specified location, in alphabetical order by department name.
- Find the average salary of employees who manage departments in a user-specified location. You can assume that no one manages more than one department.

1. Describe the file structures and indexes that you would choose.
2. You subsequently realize that updates to these relations are frequent. Because indexes incur a high overhead, can you think of a way to improve performance on these queries without using indexes?

Answer 16.9 The answer to each question is given below.

1. A heap file organization for the two relations is sufficient if we create the following indexes. For the first, a clustered B+ tree index on $\langle location, dname\rangle$ would improve performance (we cannot list the names of the managers because there is no name attribute present). We can also have a hash index on *eid* on the Emp relation to speed up the second query: we find all of the *managerids* from the B+ tree index, and then use the hash index to find their salaries.
2. Without indexes, we can use horizontal decompostion of the Dept relation based on the location. We can also try sorted file organizations, with the relation Dept sorted on *dname* and Emp on *eid*.

Exercise 16.10 For each of the following queries, identify one possible reason why an optimizer might not find a good plan. Rewrite the query so that a good plan is likely to be found. Any available indexes or known constraints are listed before each query; assume that the relation schemas are consistent with the attributes referred to in the query.

1. An index is available on the *age* attribute.

```
SELECT E.dno
FROM   Employee E
WHERE  E.age=20 OR E.age=10
```

2. A B+ tree index is available on the *age* attribute.

```
SELECT E.dno
FROM   Employee E
WHERE  E.age<20 AND E.age>10
```

3. An index is available on the *age* attribute.

```
SELECT E.dno
FROM   Employee E
WHERE  2*E.age<20
```

4. No indexes are available.

```
SELECT DISTINCT *
FROM   Employee E
```

5. No indexes are available.

```
SELECT   AVG (E.sal)
FROM     Employee E
GROUP BY E.dno
HAVING   E.dno=22
```

6. *sid* in Reserves is a foreign key that refers to Sailors.

```
SELECT   S.sid
FROM     Sailors S, Reserves R
WHERE    S.sid=R.sid
```

Answer 16.10 Answer omitted.

Exercise 16.11 Consider the following two ways of computing the names of employees who earn more than $100,000 and whose age is equal to their manager's age. First, a nested query:

```
SELECT   E1.ename
FROM     Emp E1
WHERE    E1.sal > 100 AND E1.age = ( SELECT E2.age
                                     FROM   Emp E2, Dept D2
                                     WHERE  E1.dname = D2.dname
                                            AND D2.mgr = E2.ename )
```

Second, a query that uses a view definition:

```
SELECT    E1.ename
FROM      Emp E1, MgrAge A
WHERE     E1.dname = A.dname AND E1.sal > 100 AND E1.age = A.age

CREATE VIEW MgrAge (dname, age)
       AS SELECT D.dname, E.age
          FROM   Emp E, Dept D
          WHERE  D.mgr = E.ename
```

1. Describe a situation in which the first query is likely to outperform the second query.
2. Describe a situation in which the second query is likely to outperform the first query.
3. Can you construct an equivalent query that is likely to beat both these queries when every employee who earns more than \$100,000 is either 35 or 40 years old? Explain briefly.

Answer 16.11 1. Consider the case when there are very few or no employees having salary more than 100K. Then in the first query the nested part would not be computed (due to short circuit evaluation) whereas in the second query the join of Emp and MgrAge would be computed irrespective of the number of Employees with sal > 100K.

Also, if there is an index on *dname*, then the nested portion of of the first query will be efficient. However, the index does not affect the view in the second query since it is used from a view.

2. In the case when there are a large number of employees with sal > 100K and the Dept relation is large, in the first query the join of Dept and Emp would be computed for each tuple in Emp that satisfies the condition E1.sal > 100K, whereas in the latter the join is computed only once.

3. In this case the selectivity of age may be very high. So if we have a B+ Tree index on $\langle age, sal\rangle$, then the following query may perform better.

```
SELECT    E1.ename
FROM      Emp E1
WHERE     E1.age=35 AND E1.sal > 100 AND E1.age =
          ( SELECT  E2.age
            FROM    Emp E2, Dept D2
            WHERE   E1.dname = D2.dname AND D2.mgr = E2.ename)
UNION
```

```
SELECT  E1.ename
FROM    Emp E1
WHERE   E1.age = 40 AND E1.sal > 100 AND E1.age =
        ( SELECT E2.age
        FROM     Emp E2, Dept D2
        WHERE    E1.dname = D2.dname AND D2.mgr = E2.ename)
```

17

SECURITY

Exercise 17.1 Briefly answer the following questions based on this schema:

> Emp(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`)
> Works(<u>*eid:* `integer`, *did:* `integer`</u>, *pct_time:* `integer`)
> Dept(<u>*did:* `integer`</u>, *budget:* `real`, *managerid:* `integer`)

1. Suppose you have a view SeniorEmp defined as follows:

```
CREATE VIEW SeniorEmp (sname, sage, salary)
       AS SELECT E.ename, E.age, E.salary
          FROM   Emp E
          WHERE  E.age > 50
```

   Explain what the system will do to process the following query:

```
SELECT S.sname
FROM   SeniorEmp S
WHERE  S.salary > 100,000
```

2. Give an example of a view on Emp that could be automatically updated by updating Emp.
3. Give an example of a view on Emp that would be impossible to update (automatically) and explain why your example presents the update problem that it does.
4. Consider the following view definition:

```
CREATE VIEW DInfo (did, manager, numemps, totsals)
       AS SELECT   D.did, D.managerid, COUNT (*), SUM (E.salary)
          FROM     Emp E, Works W, Dept D
          WHERE    E.eid = W.eid AND W.did = D.did
          GROUP BY D.did, D.managerid
```

(a) Give an example of a view update on DInfo that could (in principle) be implemented automatically by updating one or more of the relations Emp, Works, and Dept. Does SQL-92 allow such a view update?

(b) Give an example of a view update on DInfo that cannot (even in principle) be implemented automatically by updating one or more of the relations Emp, Works, and Dept. Explain why.

(c) How could the view DInfo help in enforcing security?

Answer 17.1 The answer to each question is given below.

1. The system will do the following:

```
SELECT   S.name
FROM     ( SELECT  E.ename AS name, E.age, E.salary
           FROM    Emp E
           WHERE   E.age > 50 ) AS S
WHERE    S.salary > 100000
```

2. The following view on Emp can be updated automatically by updating Emp:

```
CREATE VIEW SeniorEmp (eid, name, age, salary)
       AS SELECT E.eid, E.ename, E.age, E.salary
          FROM   Emp E
          WHERE  E.age > 50
```

3. The following view cannot be updated automatically because it is not clear which employee records will be affected by a given update:

```
CREATE VIEW AvgSalaryByAge (age, avgSalary)
       AS SELECT    E.eid, AVG (E.salary)
          FROM      Emp E
          GROUP BY  E.age
```

4. (a) If DInfo.manager is updated, it could, in principle, be implemented automatically by updating the Dept relation to reflect a change in the manager of department DInfo.did. However, since SQL/92 does not allow an update on a view definition based on more than one base relation, this view update is not allowed.

   (b) If DInfo.totsals is updated, this change cannot be implemented automatically at all because it is not clear which of the employees' salary fields need to be changed.

(c) Views are an important component of the security mechanisms provided by a relational DBMS. By defining views on the base relations, we can present needed information to a user while *hiding* other information that perhaps the user should not be given access to.
As an example the chairman of a company might want his secretary to be able to look at the total salaries given to a department under him, but not at the individual salaries of the employees working in those departments. This view definition would be useful in that case and provides a layer of security that prevents the secretary from viewing or changing the salaries of the employees.

Exercise 17.2 You are the DBA for the VeryFine Toy Company, and you create a relation called Employees with fields *ename*, *dept*, and *salary*. For authorization reasons, you also define views EmployeeNames (with *ename* as the only attribute) and DeptInfo with fields *dept* and *avgsalary*. The latter lists the average salary for each department.

1. Show the view definition statements for EmployeeNames and DeptInfo.
2. What privileges should be granted to a user who needs to know only average department salaries for the Toy and CS departments?
3. You want to authorize your secretary to fire people (you'll probably tell him whom to fire, but you want to be able to delegate this task), to check on who is an employee, and to check on average department salaries. What privileges should you grant?
4. Continuing with the preceding scenario, you don't want your secretary to be able to look at the salaries of individuals. Does your answer to the previous question ensure this? Be specific: Can your secretary possibly find out salaries of *some* individuals (depending on the actual set of tuples), or can your secretary always find out the salary of any individual that he wants to?
5. You want to give your secretary the authority to allow other people to read the EmployeeNames view. Show the appropriate command.
6. Your secretary defines two new views using the EmployeeNames view. The first is called AtoRNames and simply selects names that begin with a letter in the range A to R. The second is called HowManyNames and counts the number of names. You are so pleased with this achievement that you decide to give your secretary the right to insert tuples into the EmployeeNames view. Show the appropriate command, and describe what privileges your secretary has after this command is executed.
7. Your secretary allows Todd to read the EmployeeNames relation and later quits. You then revoke the secretary's privileges. What happens to Todd's privileges?

8. Give an example of a view update on the above schema that cannot be implemented through updates to Employees.

9. You decide to go on an extended vacation, and to make sure that emergencies can be handled, you want to authorize your boss Joe to read and modify the Employees relation and the EmployeeNames relation (and Joe must be able to delegate authority, of course, since he's too far up the management hierarchy to actually do any work). Show the appropriate SQL statements. Can Joe read the DeptInfo view?

10. After returning from your (wonderful) vacation, you see a note from Joe, indicating that he authorized his secretary Mike to read the Employees relation. You want to revoke Mike's `SELECT` privilege on Employees, but you don't want to revoke the rights that you gave to Joe, even temporarily. Can you do this in SQL?

11. Later you realize that Joe has been quite busy. He has defined a view called AllNames using the view EmployeeNames, defined another relation called StaffNames that he has access to (but that you can't access), and given his secretary Mike the right to read from the AllNames view. Mike has passed this right on to his friend Susan. You decide that even at the cost of annoying Joe by revoking some of his privileges, you simply have to take away Mike and Susan's rights to see your data. What `REVOKE` statement would you execute? What rights does Joe have on Employees after this statement is executed? What views are dropped as a consequence?

Answer 17.2 Answer omitted.

Exercise 17.3 Briefly answer the following questions.

1. Explain the intuition behind the two rules in the Bell-LaPadula model for mandatory access control.

2. Give an example of how covert channels can be used to defeat the Bell-LaPadula model.

3. Give an example of polyinstantiation.

4. Describe a scenario in which mandatory access controls prevent a breach of security that cannot be prevented through discretionary controls.

5. Describe a scenario in which discretionary access controls are required to enforce a security policy that cannot be enforced using only mandatory controls.

6. If a DBMS already supports discretionary and mandatory access controls, is there a need for encryption?

7. Explain the need for each of the following limits in a statistical database system:

(a) A maximum on the number of queries a user can pose.

(b) A minimum on the number of tuples involved in answering a query.

(c) A maximum on the intersection of two queries (i.e., on the number of tuples that both queries examine).

8. Explain the use of an audit trail, with special reference to a statistical database system.

9. What is the role of the DBA with respect to security?

10. What is public-key encryption? How does it differ from the encryption approach taken in the Data Encryption Standard (DES), and in what ways is it better than DES?

11. What are one-way functions, and what role do they play in public-key encryption?

12. Explain how a company offering services on the Internet could use public-key encryption to make its order-entry process secure. Describe how you would use DES encryption for the same purpose, and contrast the public-key and DES approaches.

Answer 17.3 The answer to each question is given below.

1. The *Simple Security Property* states that subjects can only interact with objects with a lesser or equal security class. This ensures subjects with low security classes from accessing high security objects. The **-Property* states that subjects can only create objects with a greater or equal security class. This prevents a high security subject from mistakenly creating an object with a low security class (which low security subjects could then access!).

2. One example of a covert channel is in statistical databases. If a malicious subject wants to find the salary of a new employee, and can issue queries to find the average salary in a department, and the total number of current employees in the depatment, then the malicious subject can calculate the new employees salary based on the increase in average salary and number of employees.

3. Say relation R contains the following values:

cid	*carname*	Security Class
1	Honda	U
1	Porsche	C
2	Toyota	C
3	Mazda	C
3	Ferrari	TS

Then subjects with security class U will see R as:

cid	*carname*	Security Class
1	Honda	U

Subjects with security class C will see R as:

cid	*carname*	Security Class
1	Honda	U
1	Porsche	C
2	Toyota	C
3	Mazda	C

Subjects with security class TS will see R as:

cid	*carname*	Security Class
1	Honda	U
1	Porsche	C
2	Toyota	C
3	Mazda	C
3	Ferrari	TS

4. Trojan horse tables are an example where discretionary access controls are not sufficient. If a malicious user creates a table and has access to the source code of some other user with privileges to other tables, then the malicious user can modify the source code to copy tuples from privileged tables to his or her non-privileged table.

5. Manditory access controls do not distinguish between people in the same clearance level so it is not possible to limit permissions to certain users within the same clearance level. Also, it is not possible to give only insert or select privileges to different users in the same level: all users in the same clearance level have select, insert, delete and update privileges.

6. Yes, especially if the data is transmitted over a network in a distributed environment. In these cases it is important to encrypt the data so people 'listening' on the wire cannot directly access the information.

7. (a) If a user can issue an unlimited number of queries, he or she can repeatedly decompose statistical information by gathering the statistics at each level (for example, at age ¿ 20, age ¿ 21, etc.).

   (b) If a malicious subject can query a database and retrieve single rows of statistical information, he or she may be able to isolate sensitive information such as maximum and minimum values.

   (c) Often the information from two queries can be combined to deduce or infer specific values. This is often the case with average and total aggregates. This can be prevented by restricting the tuple overlap between queries.

8. The *audit trail* is a log of updates with the authorization id of the user who issued the update. Since it is possible to infer information from statistical databases using repeated queries, or queries that target a common set of tuples, the DBA can use an audit trail to see which people issued these security-breaking queries.

9. The DBA creates new accounts, ensures that passwords are safe and changed often, assigns mandatory access control levels, and can analyze the audit trail to look for security breaches. They can also assit users with their discretionary permissions.

10. Public-key encryption is an encryption scheme that uses a public encryption key and a private decryption key. These keys are part of one-way functions whose inverse is very difficult to determine (which is why large prime numbers are involved in encryption algorithms...factoring is difficult!). The public key and private key are inverses which allow a user to encrypt any information, but only the person with the private key can decode the messages. DES has only one key and a specific decrypting algorithm. DES decoding can be more difficult and relies on only one key so both the sender and the receiver must know it.

11. A one-way function is a mathematical function whose inversese is very difficult to determine. These are used to determine the public and private keys, and to do the actual decoding: a message is encoding using the function and is decoded using the inverse of the function. Since the inverse is difficult to find, the code can not be broken easily.

12. An internet server could issue each user a public key with which to encrypt his or her data and send it back to the server (which holds all of the private keys). This way users cannot decode other users' messages, and even knowledge of the public key is not sufficient to decode the message. With DES, the encryption key is used both in encryption and decryption so sending keys to users is risky (anyone who intercepts the key can potentially decode the message).

18

TRANSACTION MANAGEMENT OVERVIEW

Exercise 18.1 Give brief answers to the following questions:

1. What is a transaction? In what ways is it different from an ordinary program (in a language such as C)?
2. Define these terms: *atomicity, consistency, isolation, durability, schedule, blind write, dirty read, unrepeatable read, serializable schedule, recoverable schedule, avoids-cascading-aborts schedule.*
3. Describe Strict 2PL.

Answer 18.1 The answer to each question is given below.

1. A *transaction* is an execution of a user program, and is seen by the DBMS as a series, or list, of actions. The actions that can be executed by a transaction include reads and writes of database objects, whereas actions in an ordinary program could involve user input, access to network devices, user interface drawing, etc.
2. Each term is described below.
   (a) *Atomic* transactions occur when all transaction actions are completed fully, or none are. This means there are no partial transactions (such as when half the actions complete and the other half do not).
   (b) *Consistency* involves beginning a transaction with a 'consistent' database, and finishing with a 'consitent' database. For example, in a abank database, money should never be "created" or "deleted" without an appropriate deposit or withdrawl. Every transaction should see a consistent database.
   (c) *Isolation* ensures that a transaction can run indpendently, without considering any side effects other concurrently runnings transactions might have. When a database interleves transaction actions for performance reasons, the database protects each transaction from the effects of other transactions.

(d) *Durability* defines the persistence of committed data: once a transaction commits, the data should persist in the database even if the system should crash before the data is written to main memory.

(e) A *schedule* is a series of (possibly overlapping) transactions.

(f) A *blind write* is when a transaction writes to an object without ever reading the object.

(g) A *dirty read* occurs when a transaction reads a database object that has been modified by another not-yet-committed transaction.

(h) An *unrepeatable read* occurs when a transaction is unable to read the same object value more than once, even though the transaction has not modified the value. Suppose a transaction T2 changes the value of an object A that has been read by a transaction T1 while T1 is still in progress. If T1 tries to read the value of A again, it will get a different result, even though it has not modified A.

(i) A *serializable schedule* over a set S of transactions is a schedule whose effect on any consistent database instance is identical to that of some complete serial schedule over the set of committed transactions in S.

(j) A *recoverable schedule* is one in which a transaction can commit only after all other transactions whose changes it has read have commited.

(k) A schedule that *avoids-cascading-aborts* is one in which transactons only read the changes of committed transactions. Such a schedule is not only recoverable, aborting a transaction can be accomplished without cascading the abort to other transactions.

Exercise 18.2 Consider the following actions taken by transaction $T1$ on database objects X and Y:

R(X), W(X), R(Y), W(Y)

1. Give an example of another transaction $T2$ that, if run concurrently to transaction T without some form of concurrency control, could interfere with $T1$.
2. Explain how the use of Strict 2PL would prevent interference between the two transactions.
3. Strict 2PL is used in many database systems. Give two reasons for its popularity.

Answer 18.2 Answer omitted.

Exercise 18.3 Consider a database with objects X and Y and assume that there are two transactions $T1$ and $T2$. Transaction $T1$ reads objects X and Y and then writes object X. Transaction $T2$ reads objects X and Y and then writes objects X and Y.

1. Give an example schedule with actions of transactions $T1$ and $T2$ on objects X and Y that results in a write-read conflict.

2. Give an example schedule with actions of transactions $T1$ and $T2$ on objects X and Y that results in a read-write conflict.

3. Give an example schedule with actions of transactions $T1$ and $T2$ on objects X and Y that results in a write-write conflict.

4. For each of the three schedules, show that Strict 2PL disallows the schedule.

Answer 18.3 The answer to each question is given below.

1. The following schedule results in a write-read conflict:
   T2:R(X), T2:R(Y), T2:W(X), T1:R(X) ...
   Here T1:R(X) is a dirty read.

2. The following schedule results in a read-write conflict:
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X), ...
   Now, T2 will get an unrepeatable read on X.

3. The following schedule results in a write-write conflict:
   T2:R(X), T2:R(Y), T1:R(X), T1:R(Y), T1:W(X), T2:W(X) ...
   Now, T2 has overwritten uncommitted data.

4. Strict 2PL resolves these conflicts as follows:

   (a) In S2PL, T1 could not get a shared lock on X because T2 would be holding an exclusive lock on X. Thus, T1 would have to wait until T2 was finished.

   (b) Here T1 could not get an exclusive lock on X because T2 would already be holding a shared or exclusive lock on X.

   (c) Same as above.

Exercise 18.4 Consider the following (incomplete) schedule S:

T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y)

1. Can you determine the serializability graph for this schedule? Assuming that all three transactions eventually commit, show the serializability graph.

2. For each of the following, modify S to create a complete schedule that satisfies the stated condition. If a modification is not possible, explain briefly. If it is possible, use the smallest possible number of actions (read, write, commit, or abort). You are free to add new actions anywhere in the schedule S, including in the middle.

(a) Resulting schedule avoids cascading aborts but is not recoverable.

(b) Resulting schedule is recoverable.

(c) Resulting schedule is conflict-serializable.

Answer 18.4 Answer omitted.

Exercise 18.5 Suppose that a DBMS recognizes *increment*, which increments an integer-valued object by 1, and *decrement* as actions, in addition to reads and writes. A transaction that increments an object need not know the value of the object; increment and decrement are versions of blind writes. In addition to shared and exclusive locks, two special locks are supported: An object must be locked in I mode before incrementing it and locked in D mode before decrementing it. An I lock is compatible with another I or D lock on the same object, but not with S and X locks.

1. Illustrate how the use of I and D locks can increase concurrency. (Show a schedule allowed by Strict 2PL that only uses S and X locks. Explain how the use of I and D locks can allow more actions to be interleaved, while continuing to follow Strict 2PL.)

2. Informally explain how Strict 2PL guarantees serializability even in the presence of I and D locks. (Identify which pairs of actions conflict, in the sense that their relative order can affect the result, and show that the use of S, X, I, and D locks according to Strict 2PL orders all conflicting pairs of actions to be the same as the order in some serial schedule.)

Answer 18.5 The answer to each question is given below.

1. Take the following two transactions as example:

   T1: Increment A, Decrement B, Read C;
   T2: Increment B, Decrement A, Read C

   If using only strict 2PL, all actions are versions of blind writes, they have to obtain exclusive locks on objects. Following strict 2PL, T1 gets an exclusive lock on A, if T2 now gets an exclusive lock on B, there will be a deadlock. Even if T1 is fast enough to have grabbed an exclusive lock on B first, T2 will now be blocked untill T1 finishes. This has little concurrency. If I and D locks are used, since I and D are compatible, T1 obtains an I-Lock on A, and a D-Lock on B; T2 can still obtain an I-Lock on B, a D-Lock on A; both transactions can be interleaved to allow maximum concurrency.

2. The pairs of actions which conflicts are:

   RW, WW, WR, IR, IW, DR, DW

We know that strict 2PL orders the first 3 conflicts pairs of actions to be the same as the order in some serial schedule. We can also show that even in the presence of I and D locks, strict 2PL also orders the latter 4 pairs of actions to be the same as the order in some serial schedule. Think of an I (or D)lock under these circumstances as an exclusive lock, since an I(D) lock is not compatible with S and X locks anyway (ie. can't get a S or X lock if another transaction has an I or D lock).

19

CONCURRENCY CONTROL

Exercise 19.1 1. Define these terms: *conflict-serializable schedule, view-serializable schedule, strict schedule.*

2. Describe each of the following locking protocols: *2PL, Conservative 2PL.*

3. Why must lock and unlock be atomic operations?

4. What is the phantom problem? Can it occur in a database where the set of database objects is fixed and only the values of objects can be changed?

5. Identify one difference in the timestamps assigned to restarted transactions when timestamps are used for deadlock prevention versus when timestamps are used for concurrency control.

6. State and justify the Thomas Write Rule.

Answer 19.1 The answer to each question is given below.

1. Each term is defined as follows:

    (a) A *conflict-serializable schedule* is a schedule that is conflict equivalent to some serial schedule. This means that all conflicting actions in any transaction in the conflict-serializable schedule can be ordered the same as some serial schedule.

    (b) A *view-serializable schedule* is view equivalent to some serial schedule. Two schedules, S1 and S2, are view equivalent if they follow:

        i. If Ti reads the initial value of object A in S1, it must also read the initial value of A in S2.

        ii. If Ti reads a value of A written by Tj in S1, it must also read the value of A written by Tj in S2.

        iii. For each data object A, the transaction (if any) that performs the final write on A in S1 must also perform the final write on A in S2.

(c) A *strict schedule* follows the rule that a value written by a transaction can not be read or overwritten by other transactions until T either aborts or commits.

2. $2PL$ ensures only serializable schedules are allowed. The rules are:
   - Each transaction must get a shared lock on an object before reading it
   - Each transaction must get an exclusive lock on an object before modifying it
   - Once a transaction releases a lock, it can not acquire any new locks

   $Conservative2PL$ is a variant of 2PL. Under this protocol, a transaction obtains all the locks that it will ever need when it begins, or blocks waiting for all these locks to become availabe. This scheme ensures that there will not be any deadlocks by removing one of the 4 necessary conditions for deadlock – hold while waiting.

3. Lock and unlock must be atomic operations because otherwise it may be possible for two transactions to obtain an exclusive lock on the same object, thereby destroying the principles of 2PL.

4. The *phantomproblem* is a situation where a transaction retrieves a collection of objects twice but sees different results, even though it does not modify any of these objects itself and follows the strict 2PL protocol. This problem usually arises in dynamic databases where a transaction cannot assume it has locked all objects of a given type (such as all sailors with rank 1; new sailors of rank 1 can be added by a second transaction after one transaction has locked all of the original ones).

5. When timestamps are used for deadlock prevention, a transaction that is aborted and re-started it is given the same timestamp that it had originally. When timestamps are used for concurrency control, a transaction that is aborted and restarted is given a new, larger timestamp.

6. To understand and justify Thomas' Write Rule fully, we need to give the complete context when it arises.

   To implement timestamp-based concurrency control scheme, the following regulations are made when transaction T wants to write object O:

   (a) If $TS(T) < RTS(O)$, the write action conflicts with the most recent read action of O, and T is therefore aborted and restarted.
   (b) If $TS(T) < WTS(O)$, a naive approach would be to abort T as well because its write action conflicts with the most recent write of O, and is out of timestamp order. But it turns out that we can safely ignore such previous write and process with this new write; this is called $Thomas'WriteRule$.
   (c) Otherwise, T writes O and WTS(O) is set to TS(T).

The justification is as follows: had $TS(T) < RTS(O)$, T would have been aborted and we would not have bothered to check the WTS(O). So to decide whether to abort T based on WTS(O), we can assume that $TS(T) >= RTS(O)$. If $TS(T) >= RTS(O)$ and $TS(T) < WTS(O)$, then $RTS(O) < WTS(O)$, which means the previous write occurred immediately before this planned-new-write of O and was never read by anyone, therefore the previous write can be safely ignored.

Exercise 19.2 Consider the following classes of schedules: *serializable, conflict-serializable, view-serializable, recoverable, avoids-cascading-aborts,* and *strict*. For each of the following schedules, state which of the above classes it belongs to. If you cannot decide whether a schedule belongs in a certain class based on the listed actions, explain briefly.

The actions are listed in the order they are scheduled, and prefixed with the transaction name. If a commit or abort is not shown, the schedule is incomplete; assume that abort/commit must follow all the listed actions.

1. T1:R(X), T2:R(X), T1:W(X), T2:W(X)
2. T1:W(X), T2:R(Y), T1:R(Y), T2:R(X)
3. T1:R(X), T2:R(Y), T3:W(X), T2:R(X), T1:R(Y)
4. T1:R(X), T1:R(Y), T1:W(X), T2:R(Y), T3:W(Y), T1:W(X), T2:R(Y)
5. T1:R(X), T2:W(X), T1:W(X), T2:Abort, T1:Commit
6. T1:R(X), T2:W(X), T1:W(X), T2:Commit, T1:Commit
7. T1:W(X), T2:R(X), T1:W(X), T2:Abort, T1:Commit
8. T1:W(X), T2:R(X), T1:W(X), T2:Commit, T1:Commit
9. T1:W(X), T2:R(X), T1:W(X), T2:Commit, T1:Abort
10. T2: R(X), T3:W(X), T3:Commit, T1:W(Y), T1:Commit, T2:R(Y), T2:W(Z), T2:Commit
11. T1:R(X), T2:W(X), T2:Commit, T1:W(X), T1:Commit, T3:R(X), T3:Commit
12. T1:R(X), T2:W(X), T1:W(X), T3:R(X), T1:Commit, T2:Commit, T3:Commit

Answer 19.2 Answer omitted.

Exercise 19.3 Consider the following concurrency control protocols: 2PL, Strict 2PL, Conservative 2PL, Optimistic, Timestamp without the Thomas Write Rule, Timestamp with the Thomas Write Rule, and Multiversion. For each of the schedules in

	2PL	S-2PL	C-2PL	Opt CC	TS w/o TWR	TS w/ TWR	Multiv.
1	N	N	N	N	N	N	N
2	Y	N	N	Y	Y	Y	Y
3	N	N	N	Y	N	N	Y
4	N	N	N	Y	N	N	Y
5	N	N	N	Y	N	Y	Y
6	N	N	N	N	N	Y	Y
7	N	N	N	Y	N	N	N
8	N	N	N	N	N	N	N
9	N	N	N	Y	N	N	N
10	N	N	N	N	Y	Y	Y
11	N	N	N	N	N	Y	N
12	N	N	N	N	N	Y	Y

Table 19.1

Exercise 19.2, state which of these protocols allows it, that is, allows the actions to occur in exactly the order shown.

For the timestamp-based protocols, assume that the timestamp for transaction Ti is i and that a version of the protocol that ensures recoverability is used. Further, if the Thomas Write Rule is used, show the equivalent serial schedule.

Answer 19.3 See the table 19.1.

Note the following abbreviations.
S-2PL: Strict 2PL; C-2PL: Conservative 2PL; Opt cc: Optimistic; TS W/O THR: Timestamp without Thomas Write Rule; TS With THR: Timestamp without Thomas Write Rule.

Thomas Write Rule is used in the following schedules, and the equivalent serial schedules are shown below:
5. T1:R(X), T1:W(X), T2:Abort, T1:Commit
6. T1:R(X), T1:W(X), T2:Commit, T1:Commit
11. T1:R(X), T2:Commit, T1:W(X), T2:Commit, T3:R(X), T3:Commit

Exercise 19.4 Consider the following sequences of actions, listed in the order they are submitted to the DBMS:

- **Sequence S1:** T1:R(X), T2:W(X), T2:W(Y), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

- **Sequence S2:** T1:R(X), T2:W(Y), T2:W(X), T3:W(Y), T1:W(Y), T1:Commit, T2:Commit, T3:Commit

For each sequence and for each of the following concurrency control mechanisms, describe how the concurrency control mechanism handles the sequence.

Assume that the timestamp of transaction Ti is i. For lock-based concurrency control mechanisms, add lock and unlock requests to the above sequence of actions as per the locking protocol. The DBMS processes actions in the order shown. If a transaction is blocked, assume that all of its actions are queued until it is resumed; the DBMS continues with the next action (according to the listed sequence) of an unblocked transaction.

1. Strict 2PL with timestamps used for deadlock prevention.
2. Strict 2PL with deadlock detection. (Show the waits-for graph if a deadlock cycle develops.)
3. Conservative (and strict, i.e., with locks held until end-of-transaction) 2PL.
4. Optimistic concurrency control.
5. Timestamp concurrency control with buffering of reads and writes (to ensure recoverability) and the Thomas Write Rule.
6. Multiversion concurrency control.

Answer 19.4 Answer omitted.

Exercise 19.5 For each of the following locking protocols, assuming that every transaction follows that locking protocol, state which of these desirable properties are ensured: serializability, conflict-serializability, recoverability, avoid cascading aborts.

1. Always obtain an exclusive lock before writing; hold exclusive locks until end-of-transaction. No shared locks are ever obtained.
2. In addition to (1), obtain a shared lock before reading; shared locks can be released at any time.
3. As in (2), and in addition, locking is two-phase.
4. As in (2), and in addition, all locks held until end-of-transaction.

Answer 19.5 See the table 19.2.

Exercise 19.6 The Venn diagram (from [76]) in Figure 19.1 shows the inclusions between several classes of schedules. Give one example schedule for each of the regions S1 through S12 in the diagram.

	Serializable	Conflict-serializale	Recoverable	Avoid cascading aborts
1	No	No	No	No
2	No	No	Yes	Yes
3	Yes	Yes	Yes	Yes
4	Yes	Yes	Yes	Yes

Table 19.2

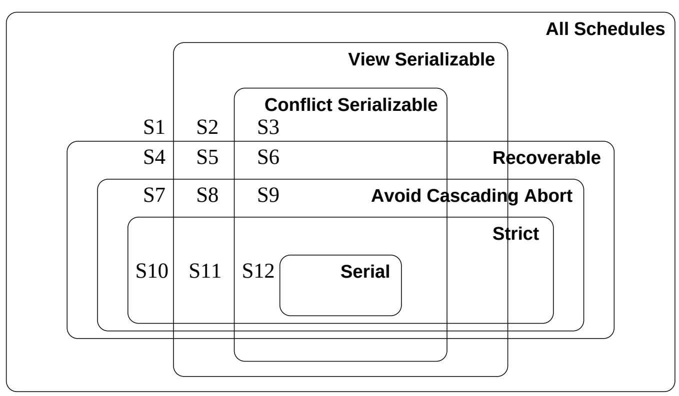

Figure 19.1 Venn Diagram for Classes of Schedules

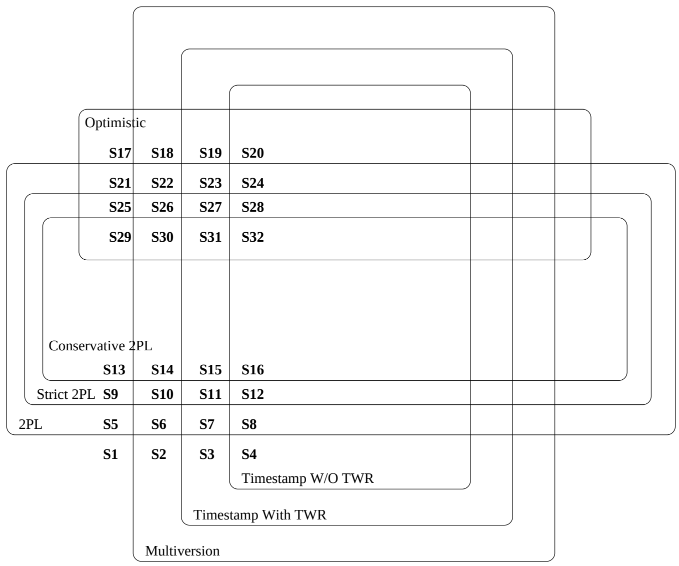

Figure 19.2

Answer 19.6 Answer omitted.

Exercise 19.7 Briefly answer the following questions:

1. Draw a Venn diagram that shows the inclusions between the classes of schedules permitted by the following concurrency control protocols: *2PL, Strict 2PL, Conservative 2PL, Optimistic, Timestamp without the Thomas Write Rule, Timestamp with the Thomas Write Rule,* and *Multiversion.*
2. Give one example schedule for each region in the diagram.
3. Extend the Venn diagram to include the class of serializable and conflict-serializable schedules.

Answer 19.7 The answer to each quesiton is given below.

1. See figure 19.2.
2. (a) Here we define the following schedule first:
    i. C1: T0:R(O),T0:Commit.
    ii. C2: T1:Begin,T2:Begin,T1:W(A),T1:Commit,T2:R(A),T2:Commit.
    iii. C3: T4:Begin,T3:Begin,T3:W(B),T3:Commit,T4:W(B),T4:Abort.
    iv. C4: T4:Begin,T3:Begin,T3:W(B),T3:Commit,T4:R(B),T4:Abort.

v. C5: T3:Begin,T4:Begin,T4:R(B),T4:Commit,T3:W(B),T3:Commit.

vi. C6: T5:Begin,T6:Begin,T6:R(D),T5:R(C),T5:Commit, T6:W(C),T6:Commit.

vii. C7: T5:Begin,T6:Begin,T6:R(D),T5:R(C),T6:W(C), T5:Commit,T6:Commit.

viii. C8: T5:Begin,T6:Begin,T5:R(C),T6:W(C),T5:R(D), T5:Commit,T6:Commit.

Then we have the following schedule for each region in the diagram.(Please note, S1: C2,C5,C8 means that S1 is the combination of schedule C2,C5,C8.)

i. S1: C2,C5,C8
ii. S2: C2,C4,C8
iii. S3: C2,C3,C8
iv. S4: C2,C8
v. S5: C2,C5,C7
vi. S6: C2,C4,C7
vii. S7: C2,C3,C7
viii. S8: C2,C7
ix. S9: C2,C5,C6
x. S10: C2,C4,C6
xi. S11: C2,C3,C6
xii. S12: C2,C6
xiii. S13: C2,C5
xiv. S14: C2,C4
xv. S15: C2,C3
xvi. S16: C2,C1

And for the rest of 16 schedules, just remove the C2 from the corresponding schedule.(eg, S17: C5,C8, which is made by removing C2 from S1.)

3. See figure 19.3.

Exercise 19.8 Answer each of the following questions briefly. The questions are based on the following relational schema:

> Emp(<u>*eid:* `integer`</u>, *ename:* `string`, *age:* `integer`, *salary:* `real`, *did:* `integer`)
> Dept(<u>*did:* `integer`</u>, *dname:* `string`, *floor:* `integer`)

and on the following update command:

replace (salary = 1.1 * EMP.salary) where EMP.ename = 'Santa'

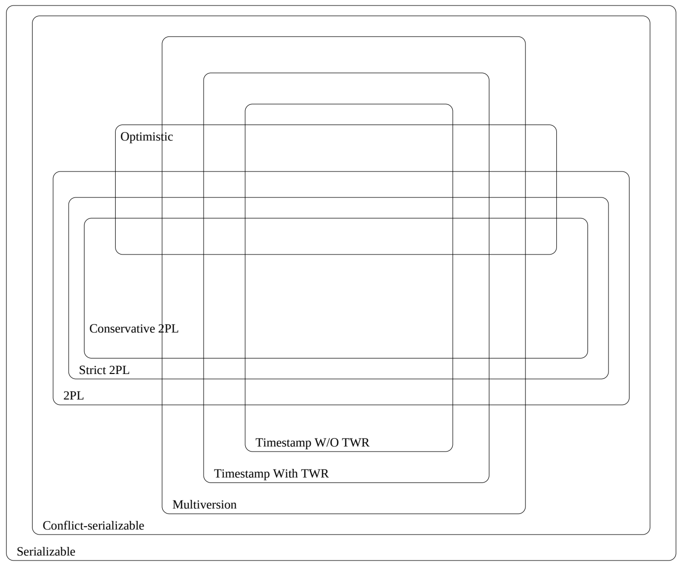

Figure 19.3

1. Give an example of a query that would conflict with this command (in a concurrency control sense) if both were run at the same time. Explain what could go wrong, and how locking tuples would solve the problem.

2. Give an example of a query or a command that would conflict with this command, such that the conflict could not be resolved by just locking individual tuples or pages, but requires index locking.

3. Explain what index locking is and how it resolves the preceding conflict.

Answer 19.8 Answer omitted.

Exercise 19.9 SQL-92 supports four isolation-levels and two access-modes, for a total of eight combinations of isolation-level and access-mode. Each combination implicitly defines a class of transactions; the following questions refer to these eight classes.

1. For each of the eight classes, describe a locking protocol that allows only transactions in this class. Does the locking protocol for a given class make any assumptions about the locking protocols used for other classes? Explain briefly.

2. Consider a schedule generated by the execution of several SQL transactions. Is it guaranteed to be conflict-serializable? to be serializable? to be recoverable?

3. Consider a schedule generated by the execution of several SQL transactions, each of which has `READ ONLY` access-mode. Is it guaranteed to be conflict-serializable? to be serializable? to be recoverable?

4. Consider a schedule generated by the execution of several SQL transactions, each of which has `SERIALIZABLE` isolation-level. Is it guaranteed to be conflict-serializable? to be serializable? to be recoverable?

5. Can you think of a timestamp-based concurrency control scheme that can support the eight classes of SQL transactions?

Answer 19.9 1. The classes `SERIALIZABLE`, `REPEATABLE READ` and `READ COMMITTED` rely on the assumption that other classes obtain exclusive locks before writing objects and hold exclusive locks until the end of the transaction.

   (a) `SERIALIZABLE` + `READ ONLY`: Strict 2PL including locks on a set of objects that it requires to be unchanged. No exclusive locks are granted.

   (b) `SERIALIZABLE` + `READ WRITE`: Strict 2PL including locks on a set of objects that it requires to be unchanged.

   (c) `REPEATABLE READ` + `READ ONLY`: Strict 2PL, only locks individual objects, not sets of objects. No exclusive locks are granted.

   (d) `REPEATABLE READ` + `READ WRITE`: Strict 2PL, only locks individual objects, not sets of objects.

   (e) `READ COMMITTED` + `READ ONLY`: Obtains shared locks before reading objects, but these locks are released immediately.

   (f) `READ COMMITTED` + `READ WRITE`: Obtains exclusive locks before writing objects, and hold these locks until the end. Obtains shared locks before reading objects, but these locks are released immediately.

   (g) `READ UNCOMMITTED` + `READ ONLY`: Do not obtain shared locks before reading objects.

   (h) `READ UNCOMMITTED` + `READ WRITE`: Obtains exclusive locks before writing objects, and hold these locks until the end. Does not obtain shared locks before reading objects.

2. Suppose we do not have any requirements for the access-mode and isolation-level of the transaction, then they are not guarenteed to be conflict-serializable, serializable, or recoverable.

3. A schedule generated by the execution of several SQL transactions, each of which having `READ ONLY` access-mode, would be guaranteed to be conflict-serializable, serializable, and recoverable. This is because the only actions are reads so there are no WW, RW, or WR conflicts.

4. A schedule generated by the execution of several SQL transactions, each of which having `SERIALIZABLE` isolation-level, would be guaranteed to be conflict-serializable, serializable, and recoverable. This is because `SERIALIZABLE` isolation level follows strict 2PL.

5. Timestamp locking with wait-die or would wait would be suitable for any `SERIALIZABLE` or `REPEATABLE READ` transaction because these follow strict 2PL. This could be modified to allow `READ COMMITTED` by allowing other transactions with a higher priority to read values changed by this transaction, as long as they didn't need to overwrite the changes. `READ UNCOMMITED` transactions can only be in the `READ ONLY` access mode, so they can read from any timestamp.

Exercise 19.10 Consider the tree shown in Figure 19.5. Describe the steps involved in executing each of the following operations according to the tree-index concurrency control algorithm discussed in Section 19.3.2, in terms of the order in which nodes are locked, unlocked, read and written. Be specific about the kind of lock obtained and answer each part independently of the others, always starting with the tree shown in Figure 19.5.

1. Search for data entry 40*.
2. Search for all data entries $k*$ with $k \leq 40$.
3. Insert data entry 62*.
4. Insert data entry 40*.
5. Insert data entries 62* and 75*.

Answer 19.10 Answer omitted.

Exercise 19.11 Consider a database that is organized in terms of the following hierarachy of objects: The database itself is an object (D), and it contains two files ($F1$ and $F2$), each of which contains 1000 pages ($P1 \ldots P1000$ and $P1001 \ldots P2000$, respectively). Each page contains 100 records, and records are identified as $p : i$, where p is the page identifier and i is the slot of the record on that page.

Multiple-granularity locking is used, with S, X, IS, IX and SIX locks, and database-level, file-level, page-level and record-level locking. For each of the following operations, indicate the sequence of lock requests that must be generated by a transaction that wants to carry out (just) these operations:

1. Read record $P1200 : 5$.
2. Read records $P1200 : 98$ through $P1205 : 2$.
3. Read all (records on all) pages in file $F1$.
4. Read pages $P500$ through $P520$.
5. Read pages $P10$ through $P980$.

6. Read all pages in $F1$ and modify about 10 pages, which can be identified only after reading $F1$.
7. Delete record $P1200 : 98$. (This is a blind write.)
8. Delete the first record from each page. (Again, these are blind writes.)
9. Delete all records.

Answer 19.11 The answer to each question is given below.

1. IS on D; IS on F2; IS on P1200; S on P1200:5.
2. IS on D; IS on F2; IS on P1200, S on 1201 through 1204, IS on P1205; S on P1200:98/99/100, S on P1205:1/2.
3. IS on D; S on F1
4. IS on D; IS on F1; S on P500 through P520.
5. IS on D; S on F1 (performance hit of locking 970 pages is likely to be higher than other blocked transactions).
6. IS and IX on D; SIX on F1.
7. IX on D; IX on F2; X on P1200.
   (Locking the whole page is not neccessary, but it would require some reorganization or compaction.)
8. IX on D; X on F1 and F2.
   (There are many ways to do this, there is a tradeoff between overhead and concurrency.)
9. IX on D; X on F1 and F2.

20

CRASH RECOVERY

Exercise 20.1 Briefly answer the following questions:

1. How does the recovery manager ensure atomicity of transactions? How does it ensure durability?
2. What is the difference between stable storage and disk?
3. What is the difference between a system crash and a media failure?
4. Explain the WAL protocol.
5. Describe the steal and no-force policies.

Answer 20.1 The answer to each question is given below.

1. The Recovery Manager ensures atomicity of transactions by undoing the actions of transactions that do not commit. It ensures durability by making sure that all actions of committed transactions survive system crashes and media failures.
2. Stable storage is guaranteed (with very high probability) to survive crashes and media failures. A disk might get corrupted or fail but the stable storage is still expected to retain whatever is stored in it. One of the ways of achieving stable storage is to store the information in a set of disks rather than in a single disk with some information duplicated so that the information is available even if one or two of the disks fail.
3. A system crash happens when the system stops functioning in a normal way or stops altogether. The Recovery Manager and other parts of the DBMS stop functioning (e.g. a core dump caused by a bus error) as opposed to media failure.

   In a media failure, the system is up and running but a particular entity of the system is not functioning. In this case, the Recovery Manager is still functioning and can start recovering from the failure while the system is still running (e.g., a disk is corrupted).

4. WAL Protocol: Whenever a change is made to a database object, the change is first recorded in the log and the log is written to stable storage before the change is written to disk.

5. If a steal policy is in effect, the changes made to an object in the buffer pool by a transaction can be written to disk before the transaction commits. This might be because some other transaction might "steal" the buffer page presently occupied by an uncommitted transaction.

   A no-force policy is in effect if, when a transaction commits, we need not ensure that all the changes it has made to objects in the buffer pool are immediately forced to disk.

Exercise 20.2 Briefly answer the following questions:

1. What are the properties required of LSNs?
2. What are the fields in an update log record? Explain the use of each field.
3. What are redoable log records?
4. What are the differences between update log records and CLRs?

Answer 20.2 Answer omitted.

Exercise 20.3 Briefly answer the following questions:

1. What are the roles of the Analysis, Redo and Undo phases in ARIES?
2. Consider the execution shown in Figure 20.1.
   (a) What is done during Analysis? (Be precise about the points at which Analysis begins and ends and describe the contents of any tables constructed in this phase.)
   (b) What is done during Redo? (Be precise about the points at which Redo begins and ends.)
   (c) What is done during Undo? (Be precise about the points at which Undo begins and ends.)

Answer 20.3 The answer to each question is given below.

1. The Analysis phase starts with the most recent begin_checkpoint record and proceeds forward in the log until the last log record. It determines
   (a) The point in the log at which to start the Redo pass

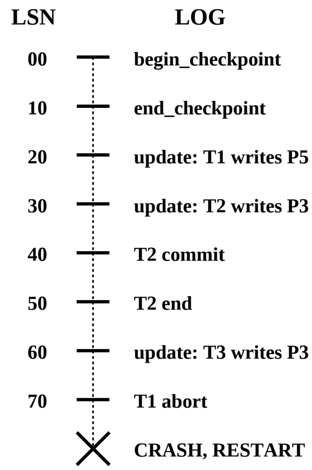

Figure 20.1 Execution with a Crash

(b) The dirty pages in the buffer pool at the time of the crash.

(c) Transactions that were active at the time of the crash which need to be undone.

The Redo phase follows Analysis and redoes all changes to any page that might have been dirty at the time of the crash. The Undo phase follows Redo and undoes the changes of all transactions that were active at the time fo the crash.

2. (a) For this example, we will assume that the Dirty Page Table and Transaction Table were empty before the start of the log. Analysis determines that the last begin_checkpoint was at LSN 00 and starts at the corresponding end_checkpoint (LSN 10).

We will denote Transaction Table records as (transID, lastLSN) and Dirty Page Table records as (pageID, recLSN) sets.

Then Analysis phase runs untill LSN 70 and does the following:

LSN 20	Adds (T1, 20) to TT and (P5, 20) to DPT
LSN 30	Adds (T2, 30) to TT and (P3, 30) to DPT
LSN 40	Changes status of T2 to "C" from "U"
LSN 50	Deletes entry for T2 from Transaction Table
LSN 60	Adds (T3, 60) to TT. Does not change P3 entry in DPT
LSN 70	Changes (T1, 20) to (T1, 70)

The final Transaction Table has two entries: (T1, 70), and (T3, 60). The final Dirty Page Table has two entries: (P5, 20), and (P3, 30).

(b) Redo Phase: Redo starts at LSN 20 (smallest recLSN in DPT).

LSN	LOG
00	update: T1 writes P2
10	update: T1 writes P1
20	update: T2 writes P5
30	update: T3 writes P3
40	T3 commit
50	update: T2 writes P5
60	update: T2 writes P3
70	T2 abort

Figure 20.2 Aborting a Transaction

LSN 20	Changes to P5 are redone.
LSN 30	P3 is retrieved and its pageLSN is checked. If the page had been written to disk before the crash (i.e. if $pageLSN >= 30$), nothing is re-done otherwise the changes are re-done.
LSN 40,50	No action
LSN 60	Changes to P3 are redone
LSN 70	No action

(c) Undo Phase: Undo starts at LSN 70 (highest lastLSN in TT). The Loser Set consists of LSNs 70 and 60. LSN 70: Adds LSN 20 to the Loser Set. Loser Set = (60, 20). LSN 60: Undoes the change on P3 and adds a CLR indicating this Undo. Loser Set = (20). LSN 20: Undoes the change on P5 and adds a CLR indicating this Undo.

Exercise 20.4 Consider the execution shown in Figure 20.2.

1. Extend the figure to show prevLSN and undonextLSN values.

2. Describe the actions taken to rollback transaction $T2$.

3. Show the log after $T2$ is rolled back, including all prevLSN and undonextLSN values in log records.

Answer 20.4 Answer omitted.

Exercise 20.5 Consider the execution shown in Figure 20.3. In addition, the system

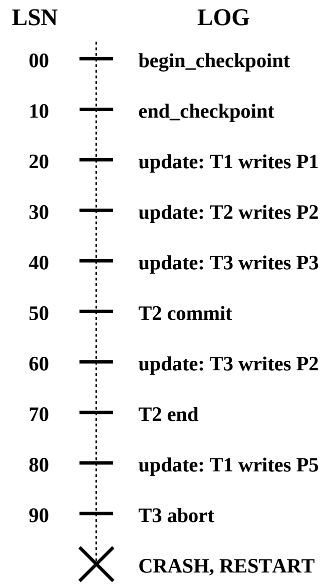

Figure 20.3 Execution with Multiple Crashes

crashes during recovery after writing two log records to stable storage and again after writing another two log records.

1. What is the value of the LSN stored in the master log record?
2. What is done during Analysis?
3. What is done during Redo?
4. What is done during Undo?
5. Show the log when recovery is complete, including all non-null prevLSN and undonextLSN values in log records.

Answer 20.5 The answer to each question is given below.

1. LSN 00 is stored in the master log record as it is the LSN of the begin_checkpoint record.
2. During analysis the following happens:

LSN 20	Add (T1,20) to TT and (P1,20) to DPT
LSN 30	Add (T2,30) to TT and (P2,30) to DPT
LSN 40	Add (T3,40) to TT and (P3,40) to DPT
LSN 50	Change status of T2 to C
LSN 60	Change (T3,40) to (T3,60)
LSN 70	Remove T2 from TT
LSN 80	Change (T1,20) to (T1,70) and add (P5,70) to DPT
LSN 90	No action

At the end of analysis, the transaction table contains the following entries: (T1,80), and (T3,60). The Dirty Page Table has the following entries: (P1,20), (P2,30), (P3,40), and (P5,80).

3. Redo starts from LSN20 (minimum recLSN in DPT).

LSN 20	Check whether P1 has pageLSN more than 10 or not. Since it is a committed transaction, we probably need not redo this update.
LSN 30	Redo the change in P2
LSN 40	Redo the change in P3
LSN 50	No action
LSN 60	Redo the changes on P2
LSN 70	No action
LSN 80	Redo the changes on P5
LSN 90	No action

4. ToUndo consists of (80, 60).

LSN 80	Undo the changes in P5. Append a CLR: Undo T1 LSN 80, set undonextLSN = 20. Add 20 to ToUndo.

ToUndo consists of (60, 20).

LSN 60	Undo the changes on P2. Append a CLR: Undo T3 LSN 60, set undonextLSN = 40. Add 40 to ToUndo.

ToUndo consists of (40, 20).

LSN 40	Undo the changes on P3. Append a CLR: Undo T3 LSN 40, T3 end

ToUndo consists of (20).

LSN 20	Undo the changes on P1. Append a CLR: Undo T1 LSN 20, T1 end

5. The log looks like the following after recovery:

LSN 00	begin_checkpoint	
LSN 10	end_checkpoint	
LSN 20	update: T1 writes P1	
LSN 30	update: T2 writes P2	
LSN 40	update: T3 writes P3	
LSN 50	T2 commit	prevLSN = 30
LSN 60	update: T3 writes P2	prevLSN = 40
LSN 70	T2 end	prevLSN = 50
LSN 80	update: T1 writes P5	prevLSN = 20
LSN 90	T3 abort	prevLSN = 60
LSN 100	CLR: Undo T1 LSN 80	undonextLSN= 20
LSN 110	CLR: Undo T3 LSN 60	undonextLSN= 40
LSN 120,125	CLR: Undo T3 LSN 40	T3 end.
LSN 130,135	CLR: Undo T1 LSN 20	T1 end.

Exercise 20.6 Briefly answer the following questions:

1. How is checkpointing done in ARIES?
2. Checkpointing can also be done as follows: Quiesce the system so that only checkpointing activity can be in progress, write out copies of all dirty pages, and include the dirty page table and transaction table in the checkpoint record. What are the pros and cons of this approach versus the checkpointing approach of ARIES?
3. What happens if a second begin_checkpoint record is encountered during the Analysis phase?
4. Can a second end_checkpoint record be encountered during the Analysis phase?
5. Why is the use of CLRs important for the use of undo actions that are not the physical inverse of the original update?
6. Give an example that illustrates how the paradigm of repeating history and the use of CLRs allow ARIES to support locks of finer granularity than a page.

Answer 20.6 Answer omitted.

Exercise 20.7 Briefly answer the following questions:

1. If the system fails repeatedly during recovery, what is the maximum number of log records that can be written (as a function of the number of update and other log records written before the crash) before restart completes successfully?

2. What is the oldest log record that we need to retain?

3. If a bounded amount of stable storage is used for the log, how can we ensure that there is always enough stable storage to hold all log records written during restart?

Answer 20.7 The answer to each question is given below.

1. Let us take the case where each log record is an update record of an uncommitted transaction and each record belongs to a different transaction. This means there are n records of n different transactions, each of which has to be undone. During recovery, we have to add a CLR record of the undone action and an end_transaction record after each CLR. Thus, we can write a maximum of $2n$ log records before restart completes.

2. The oldest begin_checkpoint referenced in any fuzzy dump or master log record.

3. One needs to ensure that there is enough to hold twice as many records as the current number of log records. If necessary, do a fuzzy dump to free up some log records whenever the number of log records goes above one third of the available space.

Exercise 20.8 Consider the three conditions under which a redo is unnecessary (Section 20.2.2).

1. Why is it cheaper to test the first two conditions?

2. Describe an execution that illustrates the use of the first condition.

3. Describe an execution that illustrates the use of the second condition.

Answer 20.8 Answer omitted.

Exercise 20.9 The description in Section 20.2.1 of the Analysis phase made the simplifying assumption that no log records appeared between the begin_checkpoint and end_checkpoint records for the most recent complete checkpoint. The following questions explore how such records should be handled.

1. Explain why log records could be written between the begin_checkpoint and end_checkpoint records.

2. Describe how the Analysis phase could be modified to handle such records.

3. Consider the execution shown in Figure 20.4. Show the contents of the end_checkpoint record.

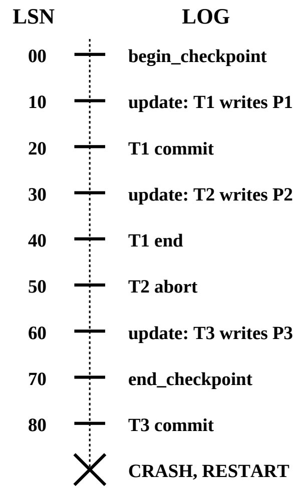

Figure 20.4 Log Records between Checkpoint Records

4. Illustrate your modified Analysis phase on the execution shown in Figure 20.4.

Answer 20.9 The answer to each question is given below.

1. In ARIES, first a begin_checkpoint record is written and then, after some time, an end_checkpoint record is written. While the end_checkpoint record is being constructed, the DBMS continues executing transactions and writing other log records. So, we could have log records between the begin_checkpoint and the end_checkpoint records. The only guarantee we have is that the transaction table and the dirty page table are accurate as of the time of the begin_checkpoint record.

2. The Analysis phase begins by examining the most recent begin_checkpoint log record and then searches for the next end_checkpoint record. Then the Dirty Page Table and the Transaction Table are initialized to the copies of those structures in the end_checkpoint. Our new Analysis phase remains the same untill here. In the old algorithm, Analysis scans the log in the forward direction until it reaches the end of the log. In the modified algorithm, Analysis goes back to the begin_checkpoint and scans the log in the forward direction.

3. The end_checkpoint record contains the transaction table and the dirty page table as of the time of the begin_checkpoint (LSN 00 in this case). Since we are assuming these tables to be empty before LSN 00, the end_checkpoint record will indicate an empty transaction table and an empty dirty page table.

4. Instead of starting from LSN 80, Analysis goes back to LSN 10 and executes as follows:

LSN 10	Add (T1,10) to TT and (P1, 10) to DPT
LSN 20	Change status of T1 from U to C.
LSN 30	Add (T2,30) to TT and (P2, 30) to DPT
LSN 40	Remove (T1,10) from TT
LSN 50	No action
LSN 60	Add (T3,60) to TT and (P3, 60) to DPT
LSN 70	No action
LSN 80	Change status of T3 from U to C.

Exercise 20.10 Answer the following questions briefly:

1. Explain how media recovery is handled in ARIES.
2. What are the pros and cons of using fuzzy dumps for media recovery?
3. What are the similarities and differences between checkpoints and fuzzy dumps?
4. Contrast ARIES with other WAL-based recovery schemes.
5. Contrast ARIES with shadow-page-based recovery.

Answer 20.10 Answer omitted.

21

PARALLEL AND DISTRIBUTED DATABASES

Exercise 21.1 Give brief answers to the following questions:

1. What are the similarities and differences between parallel and distributed database management systems?
2. Would you expect to see a parallel database built using a wide-area network? Would you expect to see a distributed database built using a wide-area network? Explain.
3. Define the terms *scale-up* and *speed-up*.
4. Why is a shared-nothing architecture attractive for parallel database systems?
5. The idea of building specialized hardware to run parallel database applications received considerable attention but has fallen out of favor. Comment on this trend.
6. What are the advantages of a distributed database management system over a centralized DBMS?
7. Briefly describe and compare the Client-Server and Collaborating Servers architectures.
8. In the Collaborating Servers architecture, when a transaction is submitted to the DBMS, briefly describe how its activities at various sites are coordinated. In particular, describe the role of transaction managers at the different sites, the concept of *subtransactions*, and the concept of *distributed transaction atomicity*.

Answer 21.1 The answer to each question is given below.

1. Parallel and distributed database management systems are similar in form, yet differ in function. The form of both types of DBMS must include direct access to both multiple storage devices and multiple processors. Note the stringency of the direct access provision. Not only does the physical architecture need to include multiple storage units and processors, but the operating system must allow the the parallel or distributed DBMS to store or process data using a particular disk or processor. An operating system that internally manages multiplicity and presents a single disk, single processor platform could not support a parallel or distributed DBMS.

   Both types of DBMS directly manipulate multiple storage devices and both have the capacity to perform operations in a non-sequential fashion, but each does so for a different functional purpose. A DBMS is made parallel primarily to improve performance by allowing non-interdependent operations to execute simultaneously. The data locations are chosen to optimize input/output requests from the processors. A DBMS is made distributed primarily to store the data in a particular location which then determines the choice of processor. Multiple locations serve as a safety net should one site fail, and provide quicker access to local data for geographically large organizations.

2. No, it would be unlikely to find a parallel database system built using a wide-area network. Any performance gains from executing operations simultaneously would surely be lost by excessive transportation costs.

   Yes, a distributed database is likely to be built using a wide-area network. Multiple copies of the data may be stored at geographically distant locations to optimize local data requests and enhance availability in the event one site fails.

3. *Speed-up* is defined as the proportional decrease in processing time due to an increase in number of disks or processors, with the amount of data held constant. In other words, for a fixed amount of data, speed-up measures how much the speed increases due to additional processors or disks.

   Scale-up is defined as the proportional increase in data processing ability due to an increase in the number of disks or processors, with the processing time held constant. In other words, in a fixed amount of time, scale-up measures the data capacity increase due to additional processors or disks.

4. The shared-nothing architecture is attractive because it allows for linear *scale-up* and *speed-up* for an arbitrary number of processors. In contrast, the shared-memory architecture can only provide these performance gains for a fixed number of processors. After a certain point, memory contention degrades performance and the gains from the shared memory approach are significantly less than those from shared-nothing alternative.

5. The production of specialized hardware requires a large capital investment which in turn requires a large market for success. While the market for database products is huge, the sub-segment of customers willing to pay a premium for parallel performance gains is smaller. Moreover, this sub-segment already has access to high-performance at a lower cost with stock hardware. Recall that using standard CPUs and interconnects in a shared nothing architecture allows for linear *speed-up* and *scale-up*. Since specialized hardware cannot provide better performance per cost, nor can it keep pace with the rapid development of stock hardware, there is a subsequent lack of development interest.

6. A distributed database system is superior to its centralized counterpart for several reasons. First, data may be replicated at multiple locations which provides increased reliability in the event that one site fails. Second, if the organization served by the DBMS is geographically diverse and access patterns are localized, storing the data locally will greatly reduce transportation costs thus improving performance. Finally, distributing data in a large organization allows for greater local autonomy so that issues of only local concern may be handled locally. A centralized database would need to coordinate too many details, e.g. ensuring no conflicts everytime someone chooses a name for a database object.

7. The client-server architecture draws a sharp distinction between the user and the data storage. The client side contains a front end for the purpose of generating queries to be sent to the server, which processes the query and responds with the data. A collaborating-servers architecture differs in that there is a collection of servers capable

of processing queries. In addition, if data is needed from multiple servers, each unit is capable of decomposing a large query into smaller queries, and sending them to the appropriate location. Thus in the collaborating-server architecture, servers not only store data and process queries, but they may also act as users of other servers.

8. A transaction submitted to a collaborating-server architecture is first evaluated to determine where the data is located and what optimized sub-queries will retrieve it most efficiently. The primary server, the recipient of the initial query, then begins a transaction and starts acquiring the necessary locks. The primary transaction manager acquires local locks in the normal fashion and issues *subtransaction* requests to the remote servers. The remote servers set about acquiring the necessary locks for the locally optimized plan and communicate back to the primary transaction manager.

   Next, the primary transaction manager waits to hear positive results from every remote transaction manager: the recovery data is stored in a safe place, the transaction commits, and executes to its entirety. If at least one remote transaction manager replies with an abort message or fails to respond at all, the primary transaction manager aborts the entire transaction. *Distributed transaction atomicity* is then guaranteed in that either the entire transaction executes everywhere or none of it executes anywhere.

Exercise 21.2 Give brief answers to the following questions:

1. Define the terms *fragmentation* and *replication*, in terms of where data is stored.
2. What is the difference between *synchronous* and *asynchronous* replication?
3. Define the term *distributed data independence*. Specifically, what does this mean with respect to querying and with respect to updating data in the presence of data fragmentation and replication?
4. Consider the *voting* and *read-one write-all* techniques for implementing synchronous replication. What are their respective pros and cons?
5. Give an overview of how asynchronous replication can be implemented. In particular, explain the terms *capture* and *apply*.
6. What is the difference between log-based and procedural approaches to implementing capture?
7. Why is giving database objects unique names more complicated in a distributed DBMS?
8. Describe a catalog organization that permits any replica (of an entire relation or a fragment) to be given a unique name and that provides the naming infrastructure required for ensuring distributed data independence.
9. If information from remote catalogs is cached at other sites, what happens if the cached information becomes outdated? How can this condition be detected and resolved?

Answer 21.2 Answer omitted.

Exercise 21.3 Consider a parallel DBMS in which each relation is stored by horizontally partitioning its tuples across all disks.

Employees(<u>*eid:* `integer`</u>, *did:* `integer`, *sal:* `real`)
Departments(<u>*did:* `integer`</u>, *mgrid:* `integer`, *budget:* `integer`)

The *mgrid* field of Departments is the *eid* of the manager. Each relation contains 20-byte tuples, and the *sal* and *budget* fields both contain uniformly distributed values in the range 0 to 1,000,000. The Employees relation contains 100,000 pages, the Departments relation contains 5,000 pages, and each processor has 100 buffer pages of 4,000 bytes each. The cost of one page I/O is t_d, and the cost of shipping one page is t_s; tuples are shipped in units of one page by waiting for a page to be filled before sending a message from processor i to processor j. There are no indexes, and all joins that are local to a processor are carried out using a sort-merge join. Assume that the relations are initially partitioned using a round-robin algorithm and that there are 10 processors.

For each of the following queries, describe the evaluation plan briefly and give its cost in terms of t_d and t_s. You should compute the total cost across all sites as well as the 'elapsed time' cost (i.e., if several operations are carried out concurrently, the time taken is the maximum over these operations).

1. Find the highest paid employee.
2. Find the highest paid employee in the department with *did* 55.
3. Find the highest paid employee over all departments with *budget* less than 100,000.
4. Find the highest paid employee over all departments with *budget* less than 300,000.
5. Find the average salary over all departments with *budget* less than 300,000.
6. Find the salaries of all managers.
7. Find the salaries of all managers who manage a department with a budget less than 300,000 and earn more than 100,000.
8. Print the *eid*s of all employees, ordered by increasing salaries. Each processor is connected to a separate printer, and the answer can appear as several sorted lists, each printed by a different processor, as long as we can obtain a fully sorted list by concatenating the printed lists (in some order).

Answer 21.3 The round-robin partitioning implies that every tuple has a equal probability of residing at each processor. Moreover, since the *sal* field of Employees and *budget* field of Departments are uniformly distributed on 0 to 1,000,000, each processor must also have a uniform distribution on this range. Also note that processing a partial page incurs the same cost as processing an entire page and the cost of writing out the result is uniformly ignored. Finally, recall that elapsed time is the maximum time taken for any one processor to complete its task.

1. Find the highest paid employee.

   Plan: Conduct a complete linear scan of the Employees relation at each processor retaining only the tuple with the highest value in *sal* field. All processors except one then send their result to a chosen processor which selects the tuple with the highest value of *sal*.

$$\begin{aligned} Total\ Cost &= (\#\ CPUs) * (Emp\ pg\ /CPU) * (I/O\ cost) \\ &+ (\#\ CPUs - 1) * (send\ cost) \\ &= (10 * 10,000 * t_d) + (9 * t_s) \\ &= 100,000 * t_d + 9 * t_s \end{aligned}$$

$$Elapsed\ Time = 10,000 * t_d + t_s$$

2. Find the highest paid employee in the department with *did* 55.

   Plan: Conduct a complete linear scan of the Employees relation at each processor retaining only the tuple with the highest value in *sal* field and a *did* field equal to 55. All processors except one then send their result to a chosen processor which selects the tuple with the highest value of *sal*.

   Total Cost: The answer is the same as for part 1 above. Even if no qualifying tuples are found at a given processor, a page should still be sent from nine processors to a chosen tenth. The page will either contain a real tuple or if a processor fails to find any tuple with *did* equal to 55, a generated tuple with *sal* equal to -1 will suffice. Note that the chosen processor must also account for the case where no tuple qualifies, simply by ignoring any tuple with *sal* equal to -1 in its final selection.

   Elapsed Time: The elapsed time is also the same as for part 1 above.

3. Find the highest paid employee over all departments with *budget* less than 100,000.

   Plan: First, conduct a complete linear scan of the Departments relation at each processor retaining only the *did* fields from tuples with *budget* less than 100,000. Recall that Departments is uniformly distributed on the *budget* field from 0 to 1,000,000, thus each processor will retain only 10% of its 500 Departments pages. Since the *did* field is 1/3 of a Departments tuple, the scan will result in approximately 16.7 pages which rounds up to 17.

   Second, each processor sends its 17 pages of retained *did* field tuples to every other processor which subsequently stores them. 10 processors send 17 pages to 9 other processors for a total of 1,530 sends. After sending, each processor has 170 (partially filled) pages of Departments tuples.

   Third, each processor joins the *did* field tuples with the Employees relation retaining only the joined tuple with the highest value in the *sal* field. Let $M = 170$ represent the number of Departments pages and $N = 10,000$ represent the number of Employees pages at each processor. Since the number of buffer pages, $100 \geq \sqrt{N}$, the refined Sort-Merge may be used for a join cost of 30,510 at each processor.

   Fourth, all processors except one then send their result to a chosen processor which selects the tuple with the highest value of *sal*.

$$\begin{aligned} Total\ Cost &= scan\ Dept\ for\ tuples\ with\ budget < 100,000 \\ &+ sending\ did\ field\ tuples\ from\ 10\ processors\ to\ 9\ others \\ &+ storing\ did\ field\ tuples\ at\ each\ processor \\ &+ joining\ with\ Emp\ and\ selecting\ max(sal)\ tuple \end{aligned}$$

$$
\begin{aligned}
& + && \textit{sending local results to the chosen processor} \\
& = && (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& && 10 * 500 * t_d \\
& && 5,000 * t_d \\
& + && (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ did\ pgs) * t_s \\
& && 10 * 9 * 17 * t_s \\
& && 1,530 * t_s \\
& + && (\#\ CPU\ storing) * (170\ did\ pgs) * (I/O\ cost) \\
& && 10 * 170 * t_d \\
& && 1,700 * t_d \\
& + && (\#\ CPU\ joining) * (join\ cost) \\
& && 10 * (3 * (170 + 10,000) * t_d) \\
& && 10 * 30,510 * t_d \\
& && 305,100 * t_d \\
& + && (\#\ CPUs - 1) * (send\ cost) \\
& && 9 * t_s \\
& = && 5,000 * t_d + 1,530 * t_s + 1,700 * t_d + 305,100 * t_d + 9 * t_s \\
& = && 311,800 * t_d + 1,539 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 500 * t_d + 153 * t_s + 170 * t_d + 30,510 * t_d + t_s \\
&= 31,180 * t_d + 154 * t_s
\end{aligned}
$$

4. Find the highest paid employee over all departments with *budget* less than 300,000.

   Plan: The evaluation of this query is identical to that in part 3 except that the probability of a Departments tuple's *budget* field being selected in step one is multiplied by three. There are then 50 pages retained by each processor and sent to every other processor for joins and maximum selection.

$$
\begin{aligned}
Total\ Cost & = && scan\ Dept\ for\ tuples\ with\ budget < 300,000 \\
& + && sending\ did\ field\ tuples\ from\ 10\ processors\ to\ 9\ others \\
& + && storing\ did\ field\ tuples\ at\ each\ processor \\
& + && joining\ with\ Emp\ and\ selecting\ max(sal)\ tuple \\
& + && sending\ local\ results\ to\ the\ chosen\ processor \\
& = && (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& && 10 * 500 * t_d \\
& && 5,000 * t_d \\
& + && (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (50\ did\ pgs) * t_s \\
& && 10 * 9 * 50 * t_s
\end{aligned}
$$

$$
\begin{aligned}
& 4,500 * t_s \\
+ \quad & (\#\ CPU\ storing) * (500\ did\ pgs) * (I/O\ cost) \\
& 10 * 500 * t_d \\
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 10 * (3 * (500 + 10,000) * t_d) \\
& 10 * 31,500 * t_d \\
& 315,000 * t_d \\
+ \quad & (\#\ CPUs\ - 1) * (send\ cost) \\
& 9 * t_s \\
= \quad & 5,000 * t_d + 4,500 * t_s + 5,000 * t_d + 315,000 * t_d + 9 * t_s \\
= \quad & 325,000 * t_d + 4,509 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 500 * t_d + 450 * t_s + 500 * t_d + 31,500 * t_d + t_s \\
&= 32,500 * t_d + 451 * t_s
\end{aligned}
$$

5. Find the average salary over all departments with *budget* less than 300,000.

   Plan: The first two steps in evaluating this query are identical to part 4. Steps three and four differ in that the desired result is an average instead of a maximum.

   First, each processor conducts a complete linear scan of the Departments relation retaining only the *did* field from tuples with a *budget* field less than 300,000. Second, each processor sends its result pages to every other processor. Third, each processor joins the *did* field tuples with the Employees relation and retains a running sum and count of the *sal* field. Fourth, each processor except one sends its sum and count to a chosen processor which divides the total sum by the total count to obtain the average. The cost is identical to part 4 above.

6. Find the salaries of all managers.

   Plan: First, conduct a complete linear scan of the Departments relation at each processor retaining only the *mgrid* field for all tuples. Since the *mgrid* field is 1/3 of each tuple, there will be 167 (rounded up) resulting pages. Second, each processor sends its result pages to every other processor which subsequently stores them. Third, each processor joins the *mgrid* field tuples with Employees thus obtaining the salaries of all managers.

$$
\begin{aligned}
Total\ Cost &= scan\ Dept\ for\ mgrid\ fields \\
&+ sending\ mgrid\ field\ tuples\ from\ 10\ processors\ to\ 9\ others \\
&+ storing\ mgrid\ field\ tuples\ at\ each\ processor \\
&+ joining\ with\ Emp \\
&= (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
&\quad 10 * 500 * t_d \\
&\quad 5,000 * t_d \\
&+ (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (167\ mgrid\ pgs) * t_s
\end{aligned}
$$

$$
\begin{aligned}
& 10 * 9 * 167 * t_s \\
& 15,030 * t_s \\
+ \quad & (\#\ CPU\ storing) * (1,670\ mgrid\ pgs) * (I/O\ cost) \\
& 10 * 1,670 * t_d \\
& 16,700 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 10 * (3 * (1,670 + 10,000) * t_d) \\
& 10 * 35,010 * t_d \\
& 350,100 * t_d \\
= \quad & 5,000 * t_d + 15,030 * t_s + 16,700 * t_d + 350,100 * t_d \\
= \quad & 386,830 * t_d + 15,030 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 500 * t_d + 1,503 * t_s + 1,670 * t_d + 35,010 * t_d \\
&= 38,683 * t_d + 1,503 * t_s
\end{aligned}
$$

7. Find the salaries of all managers who manage a department with a budget less than 300,000 and earn more than 100,000.

   Plan: The evaluation of this query is similar to that of part 6. The additional selection condition on the *budget* field is applied in step one and serves to reduce the number of pages sent and joined in steps two and three. The additional selection condition on the *sal* field is applied during the join in step three and has no effect on the final cost.

   First, conduct a complete linear scan of the Departments relation at each processor retaining only the *mgrid* field for all tuples. Since the *mgrid* field is 1/3 of each tuple and there are 150 qualifying Departments pages at each processor, there will be 50 resulting pages. Second, each processor sends its result pages to every other processor which subsequently stores them. Third, each processor joins the *mgrid* field tuples with Employees thus obtaining the salaries of all managers.

$$
\begin{aligned}
Total\ Cost \quad = \quad & scan\ Dept\ for\ mgrid\ fields \\
+ \quad & sending\ mgrid\ field\ tuples\ from\ 10\ processors\ to\ 9\ others \\
+ \quad & storing\ mgrid\ field\ tuples\ at\ each\ processor \\
+ \quad & joining\ with\ Emp \\
= \quad & (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& 10 * 500 * t_d \\
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (50\ mgrid\ pgs) * t_s \\
& 10 * 9 * 50 * t_s \\
& 4,500 * t_s \\
+ \quad & (\#\ CPU\ storing) * (500\ mgrid\ pgs) * (I/O\ cost) \\
& 10 * 500 * t_d
\end{aligned}
$$

$$
\begin{aligned}
& 5,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 10 * (3 * (500 + 10,000) * t_d) \\
& 10 * 31,500 * t_d \\
& 315,000 * t_d \\
= \quad & 5,000 * t_d + 4,500 * t_s + 5,000 * t_d + 315,000 * t_d \\
= \quad & 325,000 * t_s + 4,500 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 500 * t_d + 450 * t_s + 500 * t_d + 31,500 * t_d \\
&= 32,500 * t_d + 450 * t_s
\end{aligned}
$$

8. Print the *eids* of all employees, ordered by increasing salaries. Each processor is connected to a separate printer, and it is acceptable to have the answer in the form of several sorted lists, each printed by a different processor, as long as we can obtain a fully sorted list by concatenating the printed lists (in some order).

   Plan: At each processor, sort the Employees relation by the *sal* field and print the result. Note that the refined Sort-Merge join may be applied without the on-the-fly merge to sort at a cost of $3 * M * t_d$.

$$
\begin{aligned}
Total\ Cost &= (\#\ CPU\ sorting) * (sort\ cost) \\
&= 10 * (3 * 10,000 * t_d) \\
&= 300,000 * t_d
\end{aligned}
$$

$$
Elapsed\ Time = 30,000 * t_d
$$

Exercise 21.4 Consider the same scenario as in Exercise 21.3, except that the relations are originally partitioned using range partitioning on the *sal* and *budget* fields.

Answer 21.4 Answer omitted.

Exercise 21.5 Repeat Exercises 21.3 and 21.4 with the number of processors equal to (i) 1 and (ii) 100.

Answer 21.5 Repeat of Exercise 21.3

Recall that the round-robin distribution algorithm implies that the tuples are uniformly distributed across processors. Moreover, since the Employees and Departments relations *sal* and *budget* fields are uniformly distributed on 0 to 1,000,000, each processor must also have a uniform distribution on this range. Since elapsed time figures are redundant for the one processor case they are omitted. Also, assume for simplicity that the single processor has enough buffer pages for the Sort-Merge join algorithm, i.e., 317. Finally, for the 100 processor case, the plans are nearly identical to Exercise 20.3 and thus are also omitted.

(i) Assuming there is only 1 processor

(ii) Assuming there are 100 processors

1. Find the highest paid employee

   (i) Plan: Conduct a complete linear scan of all Employees tuples retaining only the one with the highest *sal* value.

$$Cost = (\#\ Emp\ pgs) * (I/O\ cost) = 100,000 * t_d$$

   (ii)

$$\begin{aligned} Total\ Cost &= (\#\ CPUs) * (Emp\ pgs/CPU) * (I/O\ cost) \\ &+ (\#\ CPUs - 1) * (send\ cost) \\ &= (100 * 1,000 * t_d) + (99 * t_s) \\ &= 100,000 * t_d + 99 * t_s \end{aligned}$$

$$Elapsed\ Time = 1,000 * t_d + t_s$$

2. Find the highest paid employee in the department with *did* 55.

   (i) Plan: Conduct a complete linear scan of all Employees tuples retaining only the one with the highest *sal* value and *did* field equal to 55.

$$\begin{aligned} Cost &= (\#\ Emp\ pgs) * (I/O\ cost) \\ &= 100,000 * t_d \end{aligned}$$

   (ii) Total and elapsed costs are identical to part 1 above.

3. Find the highest paid employee over all departments with *budget* less than 100,000.

   (i) Plan: join the Employees and Departments relations retaining only the one with the highest salary and budget less than 100,000.

$$\begin{aligned} Cost &= 3 * (\#\ Dept\ pgs\ + \#\ Emp\ pgs) * (I/O\ cost) \\ &= 3 * (100,000 + 5,000) * t_d \\ &= 315,000 * t_d \end{aligned}$$

   (ii)

$$\begin{aligned} Total\ Cost &= scan\ Dept\ for\ tuples\ with\ budget < 100,000 \\ &+ sending\ did\ pgs\ from\ 100\ processors\ to\ 99\ others \\ &+ storing\ did\ pgs\ at\ each\ processor \\ &+ joining\ with\ Emp\ and\ selecting\ max(sal)\ tuple \\ &+ sending\ local\ results\ to\ the\ chosen\ processor \end{aligned}$$

$$
\begin{aligned}
&= && (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& && 100 * 50 * t_d \\
& && 5,000 * t_d \\
&+ && (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (2\ did\ pgs) * t_s \\
& && 100 * 99 * 2 * t_s \\
& && 19,800 * t_s \\
&+ && (\#\ CPU\ storing) * (200\ did\ pgs) * (I/O\ cost) \\
& && 100 * 200 * t_d \\
& && 20,000 * t_d \\
&+ && (\#\ CPU\ joining) * (join\ cost) \\
& && 100 * (3 * (200 + 1,000) * t_d) \\
& && 100 * 3,600 * t_d \\
& && 360,000 * t_d \\
&+ && (\#\ CPUs - 1) * (send\ cost) \\
& && 99 * t_s \\
&= && 5,000 * t_d + 19,800 * t_s + 20,000 * t_d + 360,000 * t_d + 99 * t_s \\
&= && 385,000 * t_d + 19,899 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 198 * t_s + 200 * t_d + 3,600 * t_d + t_s \\
&= 3,850 * t_d + 199 * t_s
\end{aligned}
$$

4. Find the highest paid employee over all departments with a budget less than 300,000.
   (i) Plan: join the Employees and Departments relations retaining only the one with the highest salary and budget less than 300,000.

$$
\begin{aligned}
Cost &= 3 * (\#\ Dept\ pgs\ + \#\ Emp\ pgs) * (I/O\ cost) \\
&= 3 * (100,000 + 5,000) * t_d \\
&= 315,000 * t_d
\end{aligned}
$$

   (ii)

$$
\begin{aligned}
Total\ Cost &= && scan\ Dept\ for\ tuples\ with\ budget < 300,000 \\
&+ && sending\ did\ field\ pgs\ from\ 100\ processors\ to\ 99\ others \\
&+ && storing\ did\ field\ pgs\ at\ each\ processor \\
&+ && joining\ with\ Emp\ and\ selecting\ max(sal)\ tuple \\
&+ && sending\ local\ results\ to\ the\ chosen\ processor \\
&= && (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
& && 100 * 50 * t_d \\
& && 5,000 * t_d \\
&+ && (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (5\ did\ pgs) * t_s
\end{aligned}
$$

$$
\begin{aligned}
& 100 * 99 * 5 * t_s \\
& 49,500 * t_s \\
+ \quad & (\#\ CPU\ storing) * (500\ did\ pgs) * (I/O\ cost) \\
& 100 * 500 * t_d \\
& 50,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 100 * (3 * (500 + 1,000) * t_d) \\
& 10 * 4,500 * t_d \\
& 450,000 * t_d \\
+ \quad & (\#\ CPUs - 1) * (send\ cost) \\
& 99 * t_s \\
= \quad & 5,000 * t_d + 49,500 * t_s + 50,000 * t_d + 450,000 * t_d + 99 * t_s \\
= \quad & 495,000 * t_d + 49,599 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 495 * t_s + 500 * t_d + 4,500 * t_d + t_s \\
&= 4,950 * t_d + 496 * t_s
\end{aligned}
$$

5. Find the average salary over all departments with budget less than 300,000.

   (i) Plan: join the Employees and Departments relations retaining a running sum and count of the *sal* field for join tuples with a *budget* field less than 300,000. Divide the sum by the count to obtain the average.

$$
\begin{aligned}
Cost &= 3 * (\#\ Dept\ pgs + \#\ Emp\ pgs) * (I/O\ cost) \\
&= 3 * (100,000 + 5,000) * t_d \\
&= 315,000 * t_d
\end{aligned}
$$

   (ii) The cost is identical to part 4 above.

6. Find the salaries of all managers.

   (i) Plan: join the Employees and Departments relations at each processor retaining only those join tuples with *eid* equal to *mgrid*.

$$
\begin{aligned}
Cost &= 3 * (\#\ Dept\ pgs + \#\ Emp\ pgs) * (I/O\ cost) \\
&= 3 * (100,000 + 5,000) * t_d \\
&= 315,000 * t_d
\end{aligned}
$$

   (ii)

$$
\begin{aligned}
Total\ Cost = \quad & scan\ Dept\ for\ mgrid\ fields \\
+ \quad & sending\ mgrid\ pgs\ from\ 100\ processors\ to\ 99\ others \\
+ \quad & storing\ mgrid\ pgs\ at\ each\ processor
\end{aligned}
$$

$$
\begin{aligned}
&+ \quad joining\ with\ Emp \\
&= \quad (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
&\qquad 100 * 50 * t_d \\
&\qquad 5,000 * t_d \\
&+ \quad (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ mgrid\ pgs) * t_s \\
&\qquad 100 * 99 * 17 * t_s \\
&\qquad 168,300 * t_s \\
&+ \quad (\#\ CPU\ storing) * (170\ mgrid\ pgs) * (I/O\ cost) \\
&\qquad 100 * 170 * t_d \\
&\qquad 17,000 * t_d \\
&+ \quad (\#\ CPU\ joining) * (join\ cost) \\
&\qquad 100 * (3 * (1,700 + 1,000) t_d) \\
&\qquad 100 * 8,100 * t_d \\
&\qquad 810,000 * t_d \\
&= \quad 5,000 * t_d + 168,300 * t_s + 17,000 * t_d + 810,000 * t_d \\
&= \quad 832,000 * t_d + 168,300 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 1,683 * t_s + 170 * t_d + 8,100 * t_d \\
&= 8,320 * t_d + 1,683 * t_s
\end{aligned}
$$

7. Find the salaries of all managers who manage a department with a budget less than 300,000 and earn more than 100,000.

   (i) Plan: join the Employees and Department relations retaining only those joined tuples with *budget* < 300,000 and *sal* > 100,000.

$$
\begin{aligned}
Cost &= 3 * (\#\ Dept\ pages + \#\ Emp\ pages) * (I/O\ cost) \\
&= 3 * (100,000 + 5,000) * t_d \\
&= 315,000 * t_d
\end{aligned}
$$

   (ii)

$$
\begin{aligned}
Total\ Cost &= scan\ Dept\ for\ mgrid\ fields \\
&+ sending\ mgrid\ pgs\ from\ 100\ processors\ to\ 99\ others \\
&+ storing\ mgrid\ pgs\ at\ each\ processor \\
&+ joining\ with\ Emp \\
&= (\#\ CPU\ scanning) * (Dept\ pgs/CPU) * (I/O\ cost) \\
&\qquad 100 * 50 * t_d \\
&\qquad 5,000 * t_d \\
&+ (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (5\ mgrid\ pgs) * t_s \\
&\qquad 100 * 99 * 5 * t_s
\end{aligned}
$$

$$
\begin{aligned}
& 49,500 * t_s \\
+ \quad & (\#\ CPU\ storing) * (500\ mgrid\ pgs) * (I/O\ cost) \\
& 100 * 500 * t_d \\
& 50,000 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 100 * (3 * (500 + 1,000) t_d) \\
& 100 * 4,500 * t_d \\
& 450,000 * t_d \\
= \quad & 5,000 * t_d + 49,500 * t_s + 50,000 * t_d + 450,000 * t_d \\
= \quad & 505,000 * t_s + 49,500 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 495 * t_s + 500 * t_d + 4,500 * t_d \\
&= 5,050 * t_d + 495 * t_s
\end{aligned}
$$

8. Print the eids of all employees, ordered by increasing salaries.

   (i) Plan: sort the Employees relation using salary as a key and print the result.

$$
\begin{aligned}
Cost &= (100,000) * (sortcost) \\
&= 300,000 * t_d
\end{aligned}
$$

   (ii)

$$
\begin{aligned}
Total\ Cost &= (\#\ CPU\ sorting) * (sort\ cost) \\
&= 100 * (3 * 1,000 * t_d) \\
&= 300,000 * t_d
\end{aligned}
$$

$$
Elapsed\ Time = 3,000 * t_d
$$

Repeat of Exercise 21.4:

Recall that in Exercise 21.3 the range partitioning places tuples with either a *sal* or *budget* field between 0 and 10,000 at processor 1, between 10,001 and 20,000 at the processor 2, etc. The uniform distribution of values in *sal* and *budget* implies that there are equal numbers of tuples at each processor. Assuming 100 processors, there are 1,000 Employee tuples and 50 department tuples at each processor. Assuming there is only one processor implies partitioning is meaningless, thus the answers for part (i) are identical to those from part(i) directly above and are omitted.

The answers below assume that there are 100 processors.

1. Find the highest paid employee.

Plan: The tuple with the highest *sal* value is located at processor 100. Conduct a complete linear scan of the Employees relation there retaining the tuple with the highest value in the *sal* field.

$$\begin{aligned} Total\ Cost &= (\#\ of\ Emp\ pgs\ at\ CPU\ 100) * (I/O\ cost) \\ &= 1,000 * t_d \end{aligned}$$

$$Elapsed\ Time = 1,000 * t_d$$

2. Find the highest paid employee in the department with *did* 55.

   Plan: Since there is no guarantee that such a tuple might exist at any given processor, conduct a complete linear scan of all Employees tuples at each processor retaining the one with the highest *sal* value and *did* 55. Each processor except one should then send their result to a chosen processor which selects the tuple with the highest value in the *sal* field.

$$\begin{aligned} Total\ Cost &= (\#\ CPU\ scanning) * (\#\ of\ Emp\ pgs/CPU) * (I/O\ cost) \\ &+ (\#CPUs - 1) * (sendcost) \\ &= 100 * 1,000 * t_d \\ &+ 99 * t_s \\ &= 100,000 * t_d + 99 * t_s \end{aligned}$$

$$Elapsed\ Time = 1,000 * t_d + t_s$$

3. Find the highest paid employee over all departments with *budget* less than 100,000.

   Plan: Department tuples with a *budget* field less than 100,000 must be located at processors 1 through 10. The highest paid employees are located at the higher numbered processors, however; as in the 2. above, there is no guarantee that any processor has an Employees tuple with a particular *did* field value. So, processors 1 through 10 must conduct a complete linear scan of Departments retaining only the *did* field. The results are then sent to all processors which store and join them with the Employees relation retaining only the join tuple with the highest *sal* value. Finally, each processor except one sends the result to a chosen processor which selects the Employees tuple with the highest *sal* value.

$$\begin{aligned} Total\ Cost &= scan\ Dept\ for\ did\ fields\ at\ first\ ten\ CPUs \\ &+ sending\ did\ pgs\ from\ 10\ CPUs\ to\ 99\ CPUs \\ &+ storing\ did\ pgs\ at\ each\ processor \\ &+ joining\ did\ with\ Emp \\ &+ sending\ local\ results\ to\ chosen\ processor \\ &= (\#\ CPUs\ w/budget < 100,000) * (\#\ Dept\ pgs) * (I/Ocost) \end{aligned}$$

$$
\begin{aligned}
& & & (10\ CPUs) * (50\ pgs/CPU) * t_d \\
& & & 10 * 50 * t_d \\
& & & 500 * t_d \\
& & + \quad & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ did\ pgs) * t_s \\
& & & 10 * 99 * 17 * t_s \\
& & & 10 * 1,683 * t_s \\
& & & 16,830 * t_s \\
& & + \quad & (\#\ CPU\ storing) * (170\ did\ pgs) * (I/O\ cost) \\
& & & 100 * 170 * t_d \\
& & & 17,000 * t_d \\
& & + \quad & (\#\ CPU\ joining) * (join\ cost) \\
& & & 100 * (3 * (170 * 1,000) * t_d) \\
& & & 351,000 * t_d \\
& & + \quad & (\#\ CPUs\ -1) * (send\ cost) \\
& & & 99 * t_s \\
& & = \quad & 500 * t_d + 16,830 * t_s + 17,000 * t_d + 351,000 * t_d + 99 * t_s \\
& & = \quad & 368,500 * t_d + 16,929 * t_s \\
\\
& Elapsed\ Time & = \quad & 50 * t_d + 1,683 * t_s + 170 * t_d + 3,510 * t_d + t_s \\
& & = \quad & 3,730 * t_d + 1,684 * t_s
\end{aligned}
$$

4. Find the highest paid employee over all departments with a budget less than 300,000. Plan: The plan is identical to that for part 3 above except that now the first 30 processors must create relations of *did* fields and send them to all other processors.

$$
\begin{aligned}
Total\ Cost \quad & = \quad & & scan\ Dept\ for\ did\ fields\ at\ first\ thirty\ CPUs \\
& + \quad & & sending\ did\ field\ pgs\ from\ 30\ CPUs\ to\ 99\ CPUs \\
& + \quad & & storing\ did\ field\ pgs\ at\ each\ processor \\
& + \quad & & joining\ did\ field\ tuples\ with\ Emp\ tuples \\
& + \quad & & sending\ local\ results\ to\ chosen\ processor \\
& = \quad & & (\#\ CPUs\ w/budget < 300,000) * (\#\ Dept\ pgs) * (I/O\ cost) \\
& & & (30\ CPUs) * (50\ pgs/CPU) * t_d \\
& & & 30 * 50 * t_d \\
& & & 1,500 * t_d \\
& + \quad & & (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ did\ pgs) * t_s \\
& & & 30 * 99 * 17 * t_s \\
& & & 30 * 1,683 * t_s \\
& & & 50,490 * t_s \\
& + \quad & & (\#\ CPU\ storing) * (51\ did\ field\ pgs) * (I/O\ cost)
\end{aligned}
$$

$$
\begin{aligned}
& 100 * 51 * t_d \\
& 5,100 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 100 * (3 * (170 * 1,000) * t_d) \\
& 351,000 * t_d \\
+ \quad & (\#\ CPUs - 1) * (send\ cost) \\
& 99 * t_s \\
= \quad & 1,500 * t_d + 50,490 * t_s + 5,100 * t_d + 351,000 * t_d + 99 * t_s \\
= \quad & 357,600 * t_d + 50,589 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time &= 50 * t_d + 1,683 * t_s + 51 * t_d + 3,510 * t_d + t_s \\
&= 3,611 * t_d + 1,684 * t_s
\end{aligned}
$$

5. Find the average salary over all departments with budget less than 300,000.

   Plan: This query is similar to part 4 above. The difference is that instead of selecting the highest salary during the join and reporting to a chosen processor, each processor retains a running sum of the *sal* field and count of joined tuples. The chosen processor then computes the total sum and divides by the total count to obtain the average. Note that the costs are identical to part 4.

6. Find the salaries of all managers.

   Plan: Employees tuples with an *eid* field equal to a *mgrid* field of a Departments relation may be stored anywhere. Each processor should conduct a complete linear scan of its Departments tuples retaining only the *mgrid* field. Then, each processor sends the result to all others who subsequently store the *mgrid* relation. Next, each processor joins the *mgrid* relation with Employees retaining only the *sal* field of joined tuples.

$$
\begin{aligned}
Total\ Cost &= scan\ Dept\ for\ mgrid\ fields\ at\ all\ CPUs \\
&+ sending\ mgrid\ field\ tuples\ from\ 100\ CPUs\ to\ 99\ CPUs \\
&+ storing\ mgrid\ field\ tuples\ at\ each\ processor \\
&+ joining\ mgrid\ field\ tuples\ with\ Emp\ tuples \\
&+ sending\ local\ results\ to\ chosen\ processor \\
&= (\#\ CPUs\ scanning) * (\#\ Dept\ pgs) * (I/O\ cost) \\
&\quad 100 * 50 * t_d \\
&\quad 5,000 * t_d \\
&+ (\#\ CPU\ sending) * (\#\ CPU\ receiving) * (17\ did\ pgs) * t_s \\
&\quad 100 * 99 * 17 * t_s \\
&\quad 100 * 1,683 * t_s \\
&\quad 168,300 * t_s \\
&+ (\#\ CPU\ storing) * (1,700\ did\ field\ pgs) * (I/O\ cost) \\
&\quad 100 * 1,700 * t_d
\end{aligned}
$$

$$
\begin{aligned}
& 170,000 * t_d \\
+ \quad & (\# \ CPU \ joining) * (join \ cost) \\
& 100 * (3 * (1,700 * 1,000) * t_d) \\
& 810,000 * t_d \\
+ \quad & (\# \ CPUs - 1) * (send \ cost) \\
& 99 * t_s \\
= \quad & 5,000 * t_d + 168,300 * t_s + 170,000 * t_d + 810,000 * t_d + 99 * t_s \\
= \quad & 985,000 * t_d + 168,399 * t_s
\end{aligned}
$$

$$
\begin{aligned}
Elapsed\ Time \quad = \quad & 50 * t_d + 1,683 * t_s + 1,700 * t_d + 8,100 * t_d + t_s \\
= \quad & 9,850 * t_d + 1,684 * t_s
\end{aligned}
$$

7. Find the salaries of all managers who manage a department with a budget less than 300,000 and earn more than 100,000.

   Plan: Department tuples with a budget less than 300,000 are located at processors 1 through 30. Employees tuples with a *sal* fields greater than 100,000 are located at processors 11 through 100. Conduct a complete linear scan of all Department tuples retaining only the *mgrid* field of tuples with a *budget* field less than 300,000. Send the new *mgrid* relation to processors 11 through 100. Next, processors 11 through 100 join the new *mgrid* relation with Employees to obtain the desired result. Finally, the answer is forwarded (costlessly) to the chosen output device.

$$
\begin{aligned}
Total\ Cost \quad = \quad & scan\ Dept\ for\ mgrid\ fields\ at\ first\ thirty\ CPUs \\
+ \quad & sending\ mgrid\ field\ tuples\ from\ 10\ CPUs\ to\ 90\ CPUs \\
+ \quad & sending\ mgrid\ field\ tuples\ from\ 20\ CPUs\ to\ 89\ CPUs \\
+ \quad & storing\ mgrid\ field\ tuples\ at\ 90\ CPUs \\
+ \quad & joining\ mgrid\ field\ tuples\ with\ Emp\ tuples\ in\ 90\ CPUs \\
= \quad & (\# \ CPUs\ scanning) * (\# \ Dept\ pgs/CPU) * (I/O\ cost) \\
& 30 * 50 * t_d \\
& 1,500 * t_d \\
+ \quad & (\# \ CPU\ sending) * (\# \ CPU\ receiving) * (17\ mgrid\ pgs) * t_s \\
& 10 * 90 * 17 * t_s \\
& 10 * 1,530 * t_s \\
& 15,300 * t_s \\
+ \quad & (\# \ CPU\ sending) * (\# \ CPU\ receiving) * (17\ mgrid\ pgs) * t_s \\
& 20 * 89 * 17 * t_s \\
& 20 * 1,513 * t_s \\
& 30,260 * t_s \\
+ \quad & (\# \ CPU\ storing) * (51\ mgrid\ field\ pgs) * (I/O\ cost) \\
& 90 * 51 * t_d
\end{aligned}
$$

$$\begin{aligned}
& 4,590 * t_d \\
+ \quad & (\#\ CPU\ joining) * (join\ cost) \\
& 90 * (3 * (510 + 1,000) * t_d) \\
& 90 * 4,530 * t_d \\
& 407,700 * t_d \\
= \quad & 1,500 * t_d + 15,300 * t_s + 30,260 * t_s + 4,590 * t_d + 407,700 * t_d \\
= \quad & 413,790 * t_d + 45,560 * t_s
\end{aligned}$$

$$\begin{aligned}
Elapsed\ Time &= 50 * t_d + 1,530 * t_s + 1,513 * t_s + 51 * t_d + 4,530 * t_d \\
&= 4,631 * t_d + 3,043 * t_s
\end{aligned}$$

8. Print the eids of all employees, ordered by increasing salaries.
   Plan: Sort the Employees relation at each processor and print it out.

$$\begin{aligned}
Total\ Cost &= (\#\ CPU sorting) * (sort\ cost) \\
&= 100 * (3 * 1,000 * t_d) \\
&= 300,000 * t_d
\end{aligned}$$

$$Elapsed\ Time = 1,000\ pgs * (sort\ cost)$$

Exercise 21.6 Consider the Employees and Departments relations described in Exercise 21.3. They are now stored in a distributed DBMS with all of Employees stored at Naples and all of Departments stored at Berlin. There are no indexes on these relations. The cost of various operations is as described in Exercise 21.3. Consider the query:

```
SELECT *
FROM   Employees E, Departments D
WHERE  E.eid = D.mgrid
```

The query is posed at Delhi, and you are told that only 1 percent of employees are managers. Find the cost of answering this query using each of the following plans:

1. Compute the query at Naples by shipping Departments to Naples; then ship the result to Delhi.
2. Compute the query at Berlin by shipping Employees to Berlin; then ship the result to Delhi.
3. Compute the query at Delhi by shipping both relations to Delhi.
4. Compute the query at Naples using Bloomjoin; then ship the result to Delhi.
5. Compute the query at Berlin using Bloomjoin; then ship the result to Delhi.
6. Compute the query at Naples using Semijoin; then ship the result to Delhi.

7. Compute the query at Berlin using Semijoin; then ship the result to Delhi.

Answer 21.6 Answer omitted.

Exercise 21.7 Consider your answers in Exercise 21.6. Which plan minimizes shipping costs? Is it necessarily the cheapest plan? Which do you expect to be the cheapest?

Answer 21.7 The plan that minimizes shipping costs is computing the query in Berlin using a bloomjoin and shipping the result to Delhi. However, this is not necessarily the cheapest plan. The cheapest plan is likely to be computing the query in Berlin using a semijoin (rather than the bloomjoin), and then shipping the result to Delhi.

Exercise 21.8 Consider the Employees and Departments relations described in Exercise 21.3. They are now stored in a distributed DBMS with 10 sites. The Departments tuples are horizontally partitioned across the 10 sites by *did*, with the same number of tuples assigned to each site and with no particular order to how tuples are assigned to sites. The Employees tuples are similarly partitioned, by *sal* ranges, with $sal \leq 100,000$ assigned to the first site, $100,000 < sal \leq 200,000$ assigned to the second site, and so on. In addition, the partition $sal \leq 100,000$ is frequently accessed and infrequently updated, and it is therefore replicated at every site. No other Employees partition is replicated.

1. Describe the best plan (unless a plan is specified) and give its cost:
   (a) Compute the natural join of Employees and Departments using the strategy of shipping all fragments of the smaller relation to every site containing tuples of the larger relation.
   (b) Find the highest paid employee.
   (c) Find the highest paid employee with salary less than $100,000$.
   (d) Find the highest paid employee with salary greater than $400,000$ and less than $500,000$.
   (e) Find the highest paid employee with salary greater than $450,000$ and less than $550,000$.
   (f) Find the highest paid manager for those departments stored at the query site.
   (g) Find the highest paid manager.
2. Assuming the same data distribution, describe the sites visited and the locks obtained for the following update transactions, assuming that *synchronous* replication is used for the replication of Employees tuples with $sal \leq 100,000$:
   (a) Give employees with salary less than $100,000$ a 10 percent raise, with a maximum salary of $100,000$ (i.e., the raise cannot increase the salary to more than $100,000$).
   (b) Give all employees a 10 percent raise. The conditions of the original partitioning of Employees must still be satisfied after the update.
3. Assuming the same data distribution, describe the sites visited and the locks obtained for the following update transactions, assuming that *asynchronous* replication is used for the replication of Employees tuples with $sal \leq 100,000$.

(a) For all employees with salary less than 100,000 give them a 10 percent raise, with a maximum salary of 100,000.

(b) Give all employees a 10 percent raise. After the update is completed, the conditions of the original partitioning of Employees must still be satisfied.

Answer 21.8 Answer omitted.

Exercise 21.9 Consider the Employees and Departments relations from Exercise 21.3. You are a DBA dealing with a distributed DBMS, and you need to decide how to distribute these two relations across two sites, Manila and Nairobi. Your DBMS supports only unclustered B+ tree indexes. You have a choice between synchronous and asynchronous replication. For each of the following scenarios, describe how you would distribute them and what indexes you would build at each site. If you feel that you have insufficient information to make a decision, explain briefly.

1. Half the departments are located in Manila, and the other half are in Nairobi. Department information, including that for employees in the department, is changed only at the site where the department is located, but such changes are quite frequent. (Although the location of a department is not included in the Departments schema, this information can be obtained from another table.)
2. Half the departments are located in Manila, and the other half are in Nairobi. Department information, including that for employees in the department, is changed only at the site where the department is located, but such changes are infrequent. Finding the average salary for each department is a frequently asked query.
3. Half the departments are located in Manila, and the other half are in Nairobi. Employees tuples are frequently changed (only) at the site where the corresponding department is located, but the Departments relation is almost never changed. Finding a given employee's manager is a frequently asked query.
4. Half the employees work in Manila, and the other half work in Nairobi. Employees tuples are frequently changed (only) at the site where they work.

Answer 21.9 The answer to each question is given below.

1. Given that department information is frequently changed only at the site where it is located, horizontal fragmentation based on department location (recall that location is available in another table) will increase performance. Without knowing more about access patterns to employee data, it is impossible to say precisely what should be done with the Employees relation and what indexes would be useful. However, it is likely that given its size, a similar geographically based horizontal fragmentation of Employees along with an index on its *did* field would be useful in coordinating updates to the Departments.
2. Given that department information is infrequently changed only at the site where it is located, replication of Departments at both Manila and Nairobi will have a positive effect. There is insufficient information given to clearly decide on asynchronous vs. synchronous replication. On the one hand, infrequent updates to Departments suggests

that the accuracy gains from synchronous replication may outweigh the efficiency loss. Yet on the other hand, the most frequent query is for a departmental salary average, i.e., not for a exact number. Hence the low utility of precision suggests that semi-frequent asynchronous replication may be superior.

Depending on the access patterns of the departmental average salary queries, Employees may or may not be replicated at both sites. The sheer size of Employees suggests that replication could be very costly. Avoiding it by horizontal fragmentation on the *did* field may prove optimal overall. Even if other accesses were slower, the gains in terms of faster departmental average salary queries might tip the balance. For either replication strategy, indexes on the *did* field in both Employees and Departments would greatly enhance the speed of average salary by department queries.

3. Given that Employees tuples are frequently changed at the home site, horizontal fragmentation is very appealing. Since the results of the queries are for a single employee's manager, i.e., small result relations, there is little incentive for any replication strategy for Employees. Given that Departments almost never changes and is used for exact answer queries, synchronous replication is the best alternative. The slight loss in efficiency is easily won back in the ability to find an employees' manager immediately. The overhead of indexes on each of the key fields,*eid* and *did*, is also easily justified.
4. Given that half of the employees work in Manila, the other half work in Nairobi, and Employees' tuples are frequently changed only where they work; the obvious strategy is to horizontally fragment Employees based on worker's location (assuming this information is available in another table). Indexes on Employees' *did* field for each locations fragment would also speed the frequent accesses necessary for updating the tuples.

Exercise 21.10 Suppose that the Employees relation is stored in Madison and the tuples with $sal \leq 100,000$ are replicated at New York. Consider the following three options for lock management: all locks managed at a *single site*, say, Milwaukee; *primary copy* with Madison being the primary for Employees; and *fully distributed*. For each of the lock management options, explain what locks are set (and at which site) for the following queries. Also state which site the page is read from.

1. A query submitted at Austin wants to read a page containing Employees tuples with $sal \leq 50,000$.
2. A query submitted at Madison wants to read a page containing Employees tuples with $sal \leq 50,000$.
3. A query submitted at New York wants to read a page containing Employees tuples with $sal \leq 50,000$.

Answer 21.10 Answer omitted.

Exercise 21.11 Briefly answer the following questions:

1. Compare the relative merits of centralized and hierarchical deadlock detection in a distributed DBMS.
2. What is a *phantom deadlock*? Give an example.

3. Give an example of a distributed DBMS with three sites such that no two local waits-for graphs reveal a deadlock, yet there is a global deadlock.
4. Consider the following modification to a local waits-for graph: Add a new node T_{ext}, and for every transaction T_i that is waiting for a lock at another site, add the edge $T_i \rightarrow T_{ext}$. Also add an edge $T_{ext} \rightarrow T_i$ if a transaction executing at another site is waiting for T_i to release a lock at this site.
   (a) If there is a cycle in the modified local waits-for graph that does not involve T_{ext}, what can you conclude? If every cycle involves T_{ext}, what can you conclude?
   (b) Suppose that every site is assigned a unique integer *site-id*. Whenever the local waits-for graph suggests that there might be a global deadlock, send the local waits-for graph to the site with the next higher site-id. At that site, combine the received graph with the local waits-for graph. If this combined graph does not indicate a deadlock, ship it on to the next site, and so on, until either a deadlock is detected or we are back at the site that originated this round of deadlock detection. Is this scheme guaranteed to find a global deadlock if one exists?

Answer 21.11 The answer to each question is given below.

1. A centralized deadlock detection scheme is better for a distributed DBMS with uniform access patterns across sites since dead-locks occurring between any two sites are immediately identified. However, this benefit comes at the expense of frequent communications between the central location and every other site.

   It is often the case that access patterns are more localized, perhaps by geographic area. Since deadlocks are more likely to occur among sites with frequent communication, the hierarchical scheme will be more efficient in that it checks for deadlocks where they are most likely to occur. In other words, the hierarchical scheme expends deadlock detection efforts in correlation to their probability of occurrence, thus resulting in greater efficiency.
2. A *phantom deadlock* is defined as a falsely identified deadlock resulting from the time delay in sending local waits-for information to a central or parent site. A cycle appearing in the central or parent's global waits-for graph may in actuality have disappeared by the time the graph nodes are received and constructed. The phantom may result in some transactions being killed unnecessarily.

   For example, imagine that transaction T1 at site A is waiting for T2 at site B which is in turn waiting for T1. Then the local waits-for graphs are sent to the central detection site. Meanwhile, transaction T2 aborts for an unrelated reason and T1 no longer waits. Unfortunately for T1, the central site has identified a cycle in the global waits and chooses to kill T1!
3. Imagine three transactions T1, T2, and T3 at sites A, B, and C respectively. Suppose that T1 waits for T2 which in turn waits for T3. Comparing any two graphs in this waits-for-triangle will not reveal the global deadlock.
4. (a) A cycle in the modified waits for graph not involving T_ext clearly indicates that the deadlock is internal to the site with the graph. If every cycle involves T_ext, then there may be a multiple-site or potentially global deadlock.

   (b) The scheme is guaranteed to find a global deadlock provided that the deadlock exists prior to when the first waits-for graph is sent. If this condition is met, then the global deadlock will be uncovered before any node receives a graph containing its own nodes back.

Exercise 21.12 Timestamp-based concurrency control schemes can be used in a distributed DBMS, but we must be able to generate globally unique, monotonically increasing timestamps without a bias in favor of any one site. One approach is to assign timestamps at a single site. Another is to use the local clock time and to append the site-id. A third scheme is to use a counter at each site. Compare these three approaches.

Answer 21.12 Answer omitted.

Exercise 21.13 Consider the multiple-granularity locking protocol described in Chapter 18. In a distributed DBMS the site containing the root object in the hierarchy can become a bottleneck. You hire a database consultant who tells you to modify your protocol to allow only intention locks on the root, and to implicitly grant all possible intention locks to every transaction.

1. Explain why this modification works correctly, in that transactions continue to be able to set locks on desired parts of the hierarchy.
2. Explain how it reduces the demand upon the root.
3. Why isn't this idea included as part of the standard multiple-granularity locking protocol for a centralized DBMS?

Answer 21.13 The answer to each question is given below.

1. The consultant's suggestion of allowing only intention locks on the root works correctly because it does not prevent any transaction from obtaining a shared or exclusive lock on any sub-structure contained within the root. Recall that to obtain a shared lock, a transaction must first have an intention shared lock and to get an exclusive lock it must first have an intention exclusive lock. The only limitation resulting from the modification is that transactions wishing to modify the entire structure contained within the root must wait to obtain the necessary locks on every child of the root node.
2. The demand upon the root is reduced for two reasons. First, no transaction may greedily occupy the entire structure and must choose the relevant substructure. Second, the implicit granting of all possible intention locks to every transaction requesting access to any structure contained within the root reduces the load on the Lock Manager and the size of the waiting or fairness queue. Transactions need not wait in line for the intention locks. For these reasons, the bottleneck problem will be minimized.
3. This idea is not included as part of the Multiple-Granularity locking protocol for a centralized DBMS, or in general for a distributed DBMS, because it is a custom solution to a specific problem. The standard protocol could not predict which if any root or sub-root structures may become bottlenecks and so as is typical of standards, it opts for the general solution to the given problem.

Exercise 21.14 Briefly answer the following questions:

1. Explain the need for a commit protocol in a distributed DBMS.
2. Describe 2PC. Be sure to explain the need for force-writes.
3. Why are *ack* messages required in 2PC?
4. What are the differences between 2PC and 2PC with Presumed Abort?
5. Give an example execution sequence such that 2PC and 2PC with Presumed Abort generate an identical sequence of actions.
6. Give an example execution sequence such that 2PC and 2PC with Presumed Abort generate different sequences of actions.
7. What is the intuition behind 3PC? What are its pros and cons relative to 2PC?
8. Suppose that a site does not get any response from another site for a long time. Can the first site tell whether the connecting link has failed or the other site has failed? How is such a failure handled?
9. Suppose that the coordinator includes a list of all subordinates in the *prepare* message. If the coordinator fails after sending out either an *abort* or *commit* message, can you suggest a way for active sites to terminate this transaction without waiting for the coordinator to recover? Assume that some but not all of the *abort/commit* messages from the coordinator are lost.
10. Suppose that 2PC with Presumed Abort is used as the commit protocol. Explain how the system recovers from failure and deals with a particular transaction T in each of the following cases:
    (a) A subordinate site for T fails before receiving a *prepare* message.
    (b) A subordinate site for T fails after receiving a *prepare* message but before making a decision.
    (c) A subordinate site for T fails after receiving a *prepare* message and force-writing an abort log record but before responding to the *prepare* message.
    (d) A subordinate site for T fails after receiving a *prepare* message and force-writing a prepare log record but before responding to the *prepare* message.
    (e) A subordinate site for T fails after receiving a *prepare* message, force-writing an abort log record, and sending a *no* vote.
    (f) The coordinator site for T fails before sending a *prepare* message.
    (g) The coordinator site for T fails after sending a *prepare* message but before collecting all votes.
    (h) The coordinator site for T fails after writing an *abort* log record but before sending any further messages to its subordinates.
    (i) The coordinator site for T fails after writing a *commit* log record but before sending any further messages to its subordinates.
    (j) The coordinator site for T fails after writing an *end* log record. Is it possible for the recovery process to receive an inquiry about the status of T from a subordinate?

Answer 21.14 Answer omitted.

Exercise 21.15 Consider a heterogeneous distributed DBMS.

1. Define the terms *multidatabase system* and *gateway*.
2. Describe how queries that span multiple sites are executed in a multidatabase system. Explain the role of the gateway with respect to catalog interfaces, query optimization, and query execution.
3. Describe how transactions that update data at multiple sites are executed in a multidatabase system. Explain the role of the gateway with respect to lock management, distributed deadlock detection, Two-Phase Commit, and recovery.
4. Schemas at different sites in a multidatabase system are probably designed independently. This situation can lead to *semantic heterogeneity*; that is, units of measure may differ across sites (e.g., inches versus centimeters), relations containing essentially the same kind of information (e.g., employee salaries and ages) may have slightly different schemas, and so on. What impact does this heterogeneity have on the end user? In particular, comment on the concept of distributed data independence in such a system.

Answer 21.15 The answer to each question is given below.

1. A *multi-database system* (a.k.a. heterogeneous distributed database system) is defined as a distributed DBMS where sites operate under different DBMS packages or software. A *gateway* is defined as a communication protocol or standard used by two different DBMS packages to transmit information.
2. Queries in a *multi-database system* originate in system designated as the primary DBMS for the given query. Catalog interfaces provide the primary system with the information necessary to optimize and sub-divide the query to the sub-sites where relevant data is located. The primary system sends an optimized SQL query written in a variant of SQL that complies with the *gateway* protocol. After messages are sent back and forth to ensure compliance with locking and commit protocols, the sub-sites re-optimize their queries based on their (presumably more current) catalog information. The sub-sites then process the query and return the resulting tuples to the primary site which assembles them and presents them to the user.
3. Transactions that update data at multiple sites in a heterogeneous distributed DBMS must adhere to agreed upon locking and commit protocols just as in any DBMS with concurrency control and recovery. In any distributed system, a series of messages are associated with updates between sites to guarantee safe atomic transactions. In a heterogeneous system, communication between the different types of DBMS at different sites occurs through the *gateway*. The *gateway* provides communication channels for lock requests and responses, waits-for graphs messages, transmission of recovery logs, and the prepare, yes, no, commit, ack, and abort messages associated with two-phase commit. Given the diversity, frequency, and accuracy requirements of these essential communications it comes as no surprise that efficient gateways have not yet been successfully implemented on a wide scale.
4. The existence of semantic heterogeneity may have a profoundly confusing and/or adverse effect upon the end user. Imagine trying to understand how the average summer temperature in the Sahara desert is only 50 when the measurement is mistakenly assumed to be Fahrenheit. Or even worse, imagine investing your life savings in an security from the London stock exchange because it seems so cheap (if it were really priced in US

dollars!). Beyond trivial unit conversions, there may even be different data structures and relational schema at each of the distributed sites.

In these situations, the goal of distributed data independence, the idea that the user need not know where or how the data is stored, is obviously compromised. A sophisticated DBA could hopefully avoid the misunderstandings above by implicitly converting data to the correct local units. More generally, the DBA could create different global views to mask the underlying inconsistencies between sites. However, a large widely distributed DBMS will cross cultural boundaries and whether for humor or for sorrow, will necessarily instigate some semantic confusion.

22

INTERNET DATABASES

Exercise 22.1 Briefly answer the following questions.

1. Define the following terms and describe what they are used for: HTML, URL, CGI, server-side processing, Java Servlet, JavaBean, Java server page, HTML template, CCS, XML, DTD, XSL, semistructured data, inverted file, signature file.

2. What is CGI? What are the disadvantages of an architecture using CGI scripts?

3. What is the difference between a Web server and an application server? What funcionality do typical application servers provide?

4. When is an XML document well-formed? When is an XML document valid?

Answer 22.1 The answer to each question is given below.

1. HTML (HyperText Markup Language) is a standard for describing how a file should be displayed (as in a Web browser). HTML is used to format the display of Web pages and can include (among other things), URL links to other Web pages, images, text, or Java applets.

   URL (universal resource locator) an identifier describing where a file can be found. URLs are used to locate files on the Web and many other file access protocols.

   CGI (common gateway interface) is a protocol for communicating with processes that are running concurrently with a Web server. CGI can be used to create dynamic content on Web pages, process forms, and communicate with other application servers.

   Server-side processing is the execution of business logic at the Web server's site. This technique can be used to implement complicated business models that require processing of information on the Web server.

Java Servlets are small Java programs that interact with a Web server through a set API. Servlets are executed in their own threads and can use other features of the Java API for server-side processing.

JavaBeans are reusable software components written in Java that perform well-defined tasks and can be packaged and distributed in JAR files. By combining JavaBeans together, one can quickly and easily create larger applications.

Java server pages are another form of server-side processing that facilitate Web page formatting with ease of development. Unlike Java servlets, Java server pages do not need to be recompiled each timea change is made.

HTML templates are Web pages that include proprietary markup tags that are dynamically replaced with other HTML content during server-side processing. Cold Fusion is an example of an application that uses proprietary tags.

XML (extensible markup language) is a standard set of rules for creating files consisting of user-defined markup tags. This extensibility gives users more flexibility than HTML, and allows users to structure a document any way they wish..

DTD (document type definition) is a set of rules that allows a user to specify a set of elements, attributes, and entities in an XML document. This means that a DTD is a grammar indicating which tags are allowed, required, the order of tags, and the nesting structure.

XSL (extensible style language) is a language that defines how XML documents should be displayed.

Semistructured data is a data model that describes data that contains some regularities (structure), but is not completely structured, such as relational data.

Inverted file is an index structure that enables fast retrieval of queried documents. For each query term, there is an ordered list of documents that contain the indexed term.

Signature File is an index structure that efficiently evaluates Boolean queries. Each document is represented by a fixed length string of bits that map to possible query words. Signature files make use of hash functions to map words to bits, which allow fast boolean operations. HTML (HyperText Markup Language) is a standard for describing how a file should be displayed (as in a Web browser). HTML is used to format the display of Web pages and can include (among other things), URL links to other Web pages, images, text, or Java applets.

URL (universal resource locator) an identifier describing where a file can be found. URLs are used to locate files on the Web and many other file access protocols.

CGI (common gateway interface) is a protocol for communicating with processes that are running concurrently with a Web server. CGI can be used to create dynamic content on Web pages, process forms, and communicate with other application servers.

Server-side processing is the execution of business logic at the Web server's site. This technique can be used to implement complicated business models that require processing of information on the Web server.

Java Servlets are small Java programs that interact with a Web server through a set API. Servlets are executed in their own threads and can use other features of the Java API for server-side processing.

JavaBeans are reusable software components written in Java that perform well-defined tasks and can be packaged and distributed in JAR files. By combining JavaBeans together, one can quickly and easily create larger applications.

Java server pages are another form of server-side processing that facilitate Web page formatting with ease of development. Unlike Java servlets, Java server pages do not need to be recompiled each timea change is made.

HTML templates are Web pages that include proprietary markup tags that are dynamically replaced with other HTML content during server-side processing. Cold Fusion is an example of an application that uses proprietary tags.

XML (extensible markup language) is a standard set of rules for creating files consisting of user-defined markup tags. This extensibility gives users more flexibility than HTML, and allows users to structure a document any way they wish..

DTD (document type definition) is a set of rules that allows a user to specify a set of elements, attributes, and entities in an XML document. This means that a DTD is a grammar indicating which tags are allowed, required, the order of tags, and the nesting structure.

XSL (extensible style language) is a language that defines how XML documents should be displayed.

Semistructured data is a data model that describes data that contains some regularities (structure), but is not completely structured, such as relational data.

Inverted file is an index structure that enables fast retrieval of queried documents. For each query term, there is an ordered list of documents that contain the indexed term.

Signature File is an index structure that efficiently evaluates Boolean queries. Each document is represented by a fixed length string of bits that map to possible query words. Signature files make use of hash functions to map words to bits, which allow fast boolean operations.

2. CGI is an interface that allows a Web server to communicate with other server side applications. This gives Web servers the ability to host dynamic content. The main disadvantage of CGI is that it spawns new threads and processes for each request, which can quickly overload a Web server.

3. A Web server hosts web content that is sent to client browsers. An application server provides additional online services, often in conjunction with a Web server.

Application servers manage several existing threads or processes for incoming requests, whereas web servers must spawn new threads and processes. Application servers can provide such services as online data mining tools, banking transactions, email service, online games, etc.

4. An XML document is well-formed if it follows the structural guidelines in the XML specification. An XML document is valid if it conforms to the rules in a corresponding DTD. Note that not all XML documents need to have a corresponding DTD.

Exercise 22.2 Consider our bookstore example again. Assume that customers also want to search books by title.

1. Extend the HTML document shown in Figure 22.2 on page 646 in the book to another form that allows users to input the title of a book.
2. Write a Perl script similar to the Perl script shown in Figure 22.4 on page 648 in the book that generates dynamically an HTML page with all books that have the user-specified title.

Answer 22.2 Answer omitted.

Exercise 22.3 Consider the following description of items shown in the Eggface computer mail-order catalog.

"Eggface sells hardware and software. We sell the new Palm Pilot V for $400; its part number is 345. We also sell the IBM ThinkPad 570 for only $1999; choose part number 3784. We sell both business and entertainment software. Microsoft Office 2000 has just arrived and you can purchase the Standard Edition for only $140, part number 974. The new desktop publishing software from Adobe called InDesign is here for only $200, part 664. We carry the newest games from Blizzard software. You can start playing Diablo II for only $30, part number 12, and you can purchase Starcraft for only $10, part number 812."

1. Design an HTML document that depicts the items offered by Eggface.
2. Create a well-formed XML document that describes the contents of the Eggface catalog.
3. Create a DTD for your XML document and make sure that the document you created in the last question is valid with respect to this DTD.
4. Write an XML-QL query that lists all software items in the catalog.

5. Write an XML-QL query that lists the prices of all hardware items in the catalog.

6. Depict the catalog data in the semistructured data model as shown in Figure 22.8 on page 662 in the book.

Answer 22.3 The answer to each question is given below.

1. Answers may vary; here is one possible solution:

```
<HTML>
<HEAD><TITLE>Eggface</TITLE></HEAD>
<BODY>
Product Listing<BR>

Hardware:
<UL>
     <LI> Part:  345 <BR>
          Description:  Palm Pilot V <BR>
          Price:  $400 </LI>
     <LI> Part:  3784<BR>
          Description:  IBM ThinkPad 570<BR>
          Price:  $1999 </LI>
</UL>

Business Software:
<UL>
     <LI> Part:  974<BR>
          Description:  Microsoft Office XP Standard Edition<BR>
          Price:  $140</LI>
     <LI> Part:  664<BR>
          Description:  Adobe InDesign<BR>
          Price:  $200 </LI>
</UL>

Entertainment Software:
<UL>
     <LI> Part:  12<BR>
          Description:  Blizzard Software Diablo II<BR>
          Price:  $30</LI>
     <LI> Part:  812<BR>
          Description:  Blizzard Software Starcraft<BR>
          Price:  $10 </LI>
</UL>
</BODY>
</HTML>
```

2. Answers may vary; here is one possible solution:

```
<INVENTORY>
     <COMPANY>Eggface</COMPANY>
          <CATEGORY type='hardware'>
               <PRODUCT id='345'>
                    <NAME>Palm Pilot V</NAME>
                    <PRICE>$400</PRICE>
               </PRODUCT>
               <PRODUCT id='3784'>
                    <NAME> IBM ThinkPad 570</NAME>
                    <PRICE>$1999</PRICE>
               </PRODUCT>
          </CATEGORY>
          <CATEGORY type='software' area='business'>
               <PRODUCT id=' 974'>
                    <NAME> Microsoft Office 2000 Standard Edition </NAME>
                    <PRICE>$140</PRICE>
               </PRODUCT>
               <PRODUCT id='664'>
                    <NAME>Adobe InDesign</NAME>
                    <PRICE>$200</PRICE>
               </PRODUCT>
          </CATEGORY>
          <CATEGORY type='software' area='entertainment'>
               <PRODUCT id='12'>
                    <NAME>Blizzard Software's Diablo II</NAME>
                    <PRICE>$30</PRICE>
               </PRODUCT>
               <PRODUCT id='812'>
                    <NAME>Blizzard Software's Starcraft</NAME>
                    <PRICE>$10</PRICE>
               </PRODUCT>
          </CATEGORY>
     </INVENTORY>
```

3. Answers may vary; here is one possible solution.

```
<!DOCTYPE INVENTORY [
     <!ELEMENT INVENTORY (COMPANY,CATEGORY*)>
          <!ELEMENT COMPANY (#PCDATA)>
```

```
            <!ELEMENT CATEGORY (PRODUCT*)>
                <!ATTRLIST CATEGORY type CDATA #REQUIRED area CDATA>
                <!ELEMENT PRODUCT (NAME, PRICE)>
                    <!ATTRLIST PRODUCT id CDATA #REQUIRED>
                <!ELEMENT NAME>
                <!ELEMENT PRICE>
]>
```

4. Assuming the document is at `http://www.eggface.com/inventory.xml` the solution would be:

```
WHERE <CATEGORY type='software'>
    <PRODUCT>
        <NAME> $1 </NAME>
    </PRODUCT>
</CATEGORY> IN "www.eggface.com/inventory.xml"
CONSTRUCT <RESULT>
    <SOFTWARE>$1</SOFTWARE>
</RESULT>
```

5. Assuming the document is at `http://www.eggface.com/inventory.xml` the solution would be:

```
WHERE <CATEGORY type='hardware'>
    <PRODUCT>
        <PRICE> $p </PRICE>
    </PRODUCT>
</CATEGORY> IN "www.eggface.com/inventory.xml"
CONSTRUCT <RESULT>
    <HARDWAREPRICE>$p</HARDWAREPRICE>
</RESULT>
```

Exercise 22.4 A university database contains information about professors and the courses they teach. The university has decided to publish this information on the Web and you are in charge of the execution. You are given the following information about the contents of the database:

In the fall semester 1999, the course 'Introduction to Database Management Systems' was taught by Professor Ioannidis. The course took place Mondays and Wednesdays from 9–10 a.m. in room 101. The discussion section was held on Fridays from 9–10 a.m. Also in the fall semester 1999, the course 'Advanced Database Management

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

1. Create a well-formed XML document that contains the university database.
2. Create a DTD for your XML document. Make sure that the XML document is valid with respect to this DTD.
3. Write an XML-QL query that lists the name of all professors.
4. Describe the information in a different XML document—a document that has a different structure. Create a corresponding DTD and make sure that the document is valid. Re-formulate your XML-QL query that finds the names of all professors to work with the new DTD.

Answer 22.4 Answer omitted.

Exercise 22.5 Consider the database of the FamilyWear clothes manufacturer. FamilyWear produces three types of clothes: women's clothes, men's clothes, and children's clothes. Men can choose between polo shirts and T-shirts. Each polo shirt has a list of available colors, sizes, and a uniform price. Each T-shirt has a price, a list of available colors, and a list of available sizes. Women have the same choice of polo shirts and T-shirts as men. In addition women can choose between three types of jeans: slim fit, easy fit, and relaxed fit jeans. Each pair of jeans has a list of possible waist sizes and possible lengths. The price of a pair of jeans only depends on its type. Children can choose between T-shirts and baseball caps. Each T-shirt has a price, a list of available colors, and a list of available patterns. T-shirts for children all have the same size. Baseball caps come in three different sizes: small, medium, and large. Each item has an optional sales price that is offered on special occasions.

Design an XML DTD for FamilyWear so that FamilyWear can publish its catalog on the Web.

Answer 22.5 Here is one possible DTD:

```
<!DOCTYPE CLOTHES [
     <!ELEMENT CLOTHES (MEN, WOMEN, CHILDREN)>
     <!ELEMENT TSHIRTS (TSHIRT*)>
     <!ELEMENT TSHIRT (PRICE, SALEPRICE?, COLOR+, SIZE+)>
```

```
            <!ELEMENT PRICE (#PCDATA)>
            <!ELEMENT SALEPRICE (#PCDATA)>
            <!ELEMENT COLOR (#PCDATA)>
            <!ELEMENT SIZE (#PCDATA)>
        <!ELEMENT POLOSHIRTS (POLOSHIRT*, PRICE) >
            <!ELEMENT POLOSHIRT (SALEPRICE?, COLOR+, SIZE+)>
        <!ELEMENT JEANSET (JEANPRICE*, JEANS*)>
            <!ELEMENT JEANPRICE EMPTY>
                <!ATTRLIST JEANPRICE type (slim fit|easy fit|relaxed fit) #REQUIRED
                    price CDATA #REQUIRED>
            <!ELEMENT JEANS (LENGTH, WAIST, SALEPRICE?)>
                <!ATTRLIST JEANS type (slim fit|easy fit|relaxed fit) #REQUIRED>
                <!ELEMENT LENGTH (#PCDATA)>
                <!ELEMENT WAIST (#PCDATA)>
        <!ELEMENT CHILDTSHIRTS (SIZE, CHILDTSHIRT*)>
            <!ELEMENT CHILDTSHIRT (PRICE, SALEPRICE?, COLOR+, PATTERN+)>
                <!ELEMENT PATTERN (#PCDATA)>
]>
```

Exercise 22.6 Assume you are given a document database that contains six documents. After stemming, the documents contain the following terms:

Document	Terms
1	car manufacturer Honda auto
2	auto computer navigation
3	Honda navigation
4	manufacturer computer IBM
5	IBM personal computer
6	car Beetle VW

Answer the following questions.

1. Discuss the advantages and disadvantages of inverted files versus signature files.

2. Show the result of creating an inverted file on the documents.

3. Show the result of creating a signature file with a width of 5 bits. Construct your own hashing function that maps terms to bit positions.

4. Evaluate the following queries using the inverted file and the signature file that you created: ‘car’, ‘IBM’ `AND` ‘computer’, ‘IBM’ `AND` ‘car’, ‘IBM’ `OR` ‘auto’, and ‘IBM’ `AND` ‘computer’ `AND` ‘manufacturer’.

5. Assume that the query load against the document database consists of exactly the queries that were stated in the previous question. Also assume that each of these queries is evaluated exactly once.
   (a) Design a signature file with a width of 3 bits and design a hashing function that minimizes the overall number of false positives retrieved when evaluating the
   (b) Design a signature file with a width of 6 bits and a hashing function that minimizes the overall number of false positives.
   (c) Assume you want to construct a signature file. What is the smallest signature width that allows you to evaluate all queries without retrieving any false positives?

Answer 22.6 Answer omitted.

Exercise 22.7 Assume that the base set of the HITS algorithm consists of the set of Web pages displayed in the following table. An entry should be interpreted as follows: Web page 1 has hyperlinks to pages 5 and 6.

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

Run five iterations of the HITS algorithm and find the highest ranked authority and the highest ranked hub.

Answer 22.7 Each update of the authority and hub weight vectors will be done with the previous steps values (i.e. no intermediate results will be used).

```
a0 = <1,1,1,1,1,1,1,1>
h0 = <1,1,1,1,1,1,1,1>

a1 = <3,2,1,1,2,2,2,1>
h1 = <3,2,2,0,2,2,2,1>

a2 = <6,4,2,1,5,5,5,2>
h2 = <6,4,3,0,5,4,5,1>
```

```
a3 = <14,10,4,1,10,9,9,3>
h3 = <15,10,7,0,10,8,10,1>

a4 = <28,20,8,1,25,22,22,7>
h4 = <28,19,12,0,24,18,24,1>

a5 = <66,48,18,1,47,40,40,12>
h5 = <69,47,29,0,48,36,48,1>
```

So the highest ranking authority is page 1, and the highest ranking hub is also page 1.