Database Systems: An Application-Oriented Approach (Introductory Version)

Practice Problems and Solutions for Students

Michael Kifer, Arthur Bernstein, Philip M. Lewis

Contents

Problems for Chapter 3: The Relational Data Model

1. Define the database concepts: primary key, candidate key and superkey. Is a superkey always a key? Explain.

   Solution:

   Each relation must have a primary key, which is an attribute or set of attributes used to uniquely identify tuples in the relation. Thus any legal instance of a relation **R** cannot contain distinct tuples which agree on the value of the primary key but not on the remaining set of attributes. Every tuple in a relation has a unique value for the primary key.
   Superkey is any set of attributes which satisfies the uniqueness condition
   Primary key (or candidate key) a minimal superkey; that is, no proper subset of the key is a superkey. primary key one key must be designated as the primary key.

2. Answer the following questions:

   (a) Why are keys important?

   (b) Identify which ones of the following constraints are (i) structural, (ii) semantic, (iii) static, (iv) dynamic.

      i. A class start time must be before its end time.

      ii. A table of students should contain no more than 200 rows.

      iii. The mileage of a car can not decrease.

      iv. In teaching assignment table that assigns professors to courses, each individual assignment should correspond to exactly one professor and one course.

Solution:

(a) A key uniquely identifies an entity and is minimal. The first is so that one can talk about an entity and others know which one exactly he is talking about. The second is for convenience and efficiency of identifying an entity.

(b) i. semantic, static.

ii. semantic, static.

iii. semantic, dynamic.

iv. structural, static. (This is a foreign key constraint.)

3. For a simple BBS (Bulletin Board System) we use the following SQL statements to create two tables: one storing all posted messages and the other users who can post them.

```
CREATE TABLE MESSAGE (
     mesgid INTEGER,
     poster INTEGER,
     subject CHAR(50),
     body    CHAR(255),
     postdate DATETIME,
     PRIMARY KEY mesgid,
     FOREIGN KEY poster REFERENCES USER (userid)
     ON DELETE CASCADE
     ON UPDATE CASCADE
)

CREATE TABLE USER (
     userid CHAR(50),
     password CHAR(50),
     email CHAR(50),
     status CHAR(1),
     PRIMARY KEY(userid)
)
```

(a) There is an error in one of the above statements. Point out the error, explain why it is wrong and correct the error by rewriting that SQL statement.

(b) Suppose there is a user with `userid` John in the database who has posted 100 messages. What will the DBMS do if we delete John from table USER?
What if we change John's `userid` to Michael?

(c) Write an SQL statement to create a view of those messages with all their attributes that are posted by 'John'.

(d) Write an SQL statement to create a domain such that the status attribute can only take two values, i.e., 'j' and 's'.

(e) Suppose occasionally the system will post some announcement messages, but unfortunately the system is not a user (thus it does not appear in the USER table). How can you allow these messages being posted while not adding a "system user" and not violating the foreign key constraint?

(f) One desirable advanced feature of the BBS system is that each user can post messages not only to the public, but also to a subset of other users that are explicitly specified by `userid` when the message is posted. How would you change the definitions of the above two tables so that this new feature can be implemented? (You may introduce other tables if necessary.)

Solution:

(a) In the table MESSAGE, poster is of type INTEGER and it references `userid` in USER which is of type CHAR(50). The poster column type in the table MESSAGE should be changed to CHAR(50) for it to be consistent.
Also, the `userid` in USER could be changed to INTEGER. (theoretically still correct)

(b) As specified in CREATE TABLE MESSAGE, if we delete John from the table USER, all messages with the poster = 'John' will be deleted.
When John's user id updated, tuples in the MESSAGE table for which the poster attribute has the value 'John', will be updated to John's new `userid` i.e. Michael.

(c)
```
CREATE VIEW JOHNSMESSAGE AS
SELECT * FROM MESSAGE WHERE poster = 'John';
```

(d)
```
CREATE DOMAIN StatusValue CHAR(1)
CHECK ( VALUE IN ('s', j') );
```

(e) SQL allows foreign keys to have null values. So `poster` can be null in MESSAGE. Therefore, the current structure already supports the requested feature.

(f) Introduce a new table say POSTMESSAGETO. This table has the following structure:

```
CREATE TABLE POSTMESSAGETO (
mesgid INTEGER,
receiver CHAR(50),
FOREIGN KEY poster REFERENCES USER (userid),
FOREIGN KEY receiver REFERENCES USER (userid)
)
```

Now for every message there is a list of users to whom the message is to be sent, which is maintained in the POSTMESSAGETO table.

4. Suppose you are not allowed to use the FOREIGN KEY clause. How can you express foreign-key constraints?

   For example, suppose we have a table PROFESSOR with attributes `Id`, `name`, etc., where `Id` is the primary key and a table TEACHES with attributes `ProfId`, `CrsCode`, etc. How would you specify the constraint that each professor teaches exactly one course?

   Solution:

   Use the following SQL assertion:

```
CREATE ASSERTION EachProfTeachExactlyOneCourse
  CHECK (NOT EXISTS
      (SELECT * FROM  PROFESSOR  P
       WHERE (SELECT COUNT(*) FROM  TEACHES  A
              WHERE P.Id = T.ProfId) <> 1))
```

Problems for Chapter 4: Conceptual Modeling of Databases with E-R and UML

1. An Internet store sends emails to customers and would like to use a database to keep track of which messages are sent to which customers. A message has a message id (`mId`), a subject (`subject`), and a body (`body`). A customer is identified by the email address (`email`), and customer's data includes the attributes `name`, `sex`, `householdSize`, and `address`. When an email is sent, the date (`sendDate`) is recorded. You can assume any reasonable domains for these attributes.

   (a) Give an E-R diagram that completely describes the entities and relationships of this plan.

   (b) Translate the E-R diagram into SQL tables using SQL DDL, by specifying a table for messages, a table for customers, and a table for which messages are sent to which customers, in SQL DDL.

   (c) How to specify that the subject of a message must not be longer than 60 characters, in SQL?

   (d) How to require that customers at the same address all have different names, in SQL?

   (e) How to enforce the restriction that the only valid values of sex are male or female, in SQL?

   (f) How to specify that each message must be sent to no more than one customer, in SQL?

   (g) How to specify that each customer must be sent one or more messages, in SQL?

   (h) How to specify that each message must be sent to exactly one customer, in SQL?

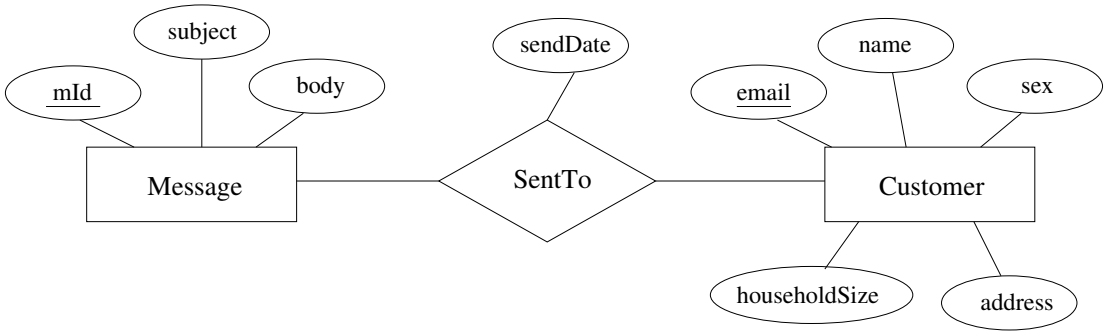
   (i) How to specify that each customer must be sent exactly one message, in SQL?

   (j) How to specify that a message can be deleted only if no customer has been sent that message, in SQL?

   (k) Shipping department needs not be concerned with the sex and household size of the customers. Create an element of an external schema which is a view of customers that does not contain these attributes.

Solution:

(a) The diagram looks something like below.



(b) The corresponding tables are shown below.

```
CREATE TABLE Message (
  mId INTEGER,
  subject CHAR(40),
  body CHAR(10000),
  PRIMARY KEY(mId))

CREATE TABLE SentTo (
  mId INTEGER,
  email CHAR(40),
  sendDate DATE NOT NULL,
  FOREIGN KEY mId REFERENCE Message,
  FOREIGN KEY email REFERENCE Customer,
  PRIMARY KEY (mId, email))

CREATE TABLE Customer (
    email CHAR(40),
    name CHAR(40),
    sex CHAR(1),
    householdSize INTEGER,
    address CHAR(50)
  PRIMARY KEY(email))
```

(c) Change the domain of `subject` to: CHAR(60)

(d) Add to the table Customer: `UNIQUE (name, address)`

(e) Add to the table Customer:

```
CHECK (sex IN ('M', 'F'))
```

Alternatively, create the following domein

```
CREATE DOMAIN Sex CHAR(1)
       CHECK (VALUE IN ('M', 'F'))
```

and change the domain of the attribute `sex` to: `Sex`

(f) Change the primary key of the Send table to `mId`. Since email will no longer be part of the primary key, it needs a NOT NULL constraint.

(g)
```
CREATE ASSERTION NoNeglectedCustomer
  CHECK (NOT EXISTS (
      SELECT * FROM Customer C
      WHERE NOT EXISTS (
          SELECT * FROM SentTo S
          WHERE C.email = S.email )))
```

(h) Add to the table MESSAGE: FOREIGN KEY `mId` REFERENCE SEND.
In the table SEND: make `mId` into the primary key and add NOT NULL to `email`.

(i) Make `email` into the primary key of SEND and add NOT NULL to `mId` (since it is no longer part of the primary key).
Add to the table CUSTOMER: `FOREIGN KEY email REFERENCE` SEND.

(j) Modify the foreign key `mId` in the table SEND by adding ON DELETE NO ACTION (this is actually the default anyway).

(k)
```
CREATE VIEW SHIPPING (email,name,address) AS
SELECT email,name,address
FROM CUSTOMER
```

2. Professor Smith would like to assign m projects to n students in a database class. Each project can be described by its name, description, deadline and status (completed or in progress); each student has a student id, a name, and an email. Students can work in groups of several persons on one of the m projects. Different groups will take different projects (assume more projects than students, so some projects will have no students assigned) and each student participates in exactly one group. After the project for a group is finished, a grade for the project is determined and given to all students in the group. Assume each group is identified by a unique group name. In the following, you are asked to help Dr. Smith design a database system to facilitate the assignment and grading of the projects,

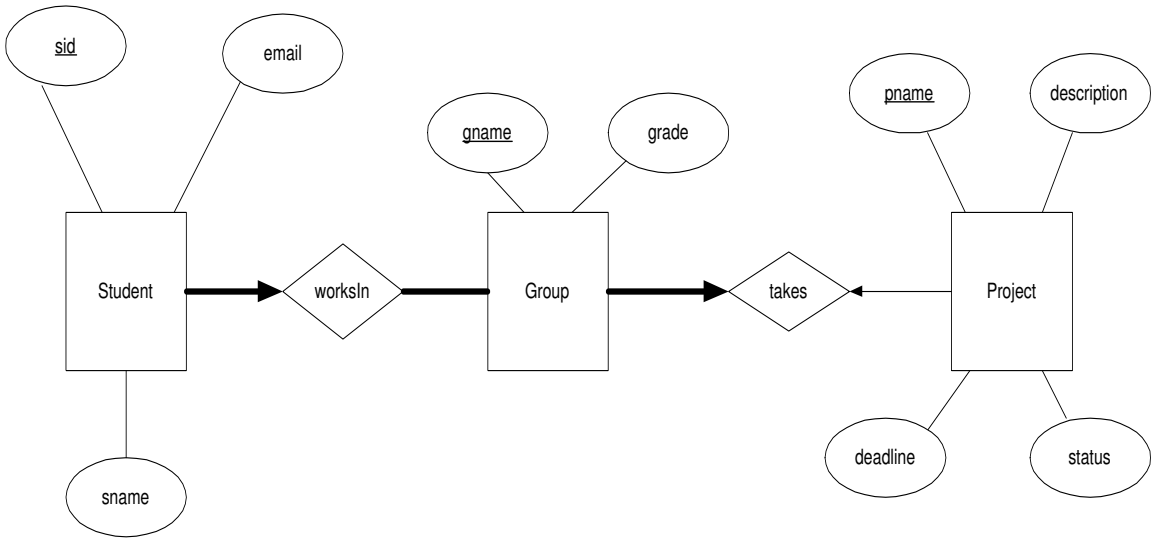
   (a) Draw an E-R diagram for the system, in particular, use arrows or thick lines to represent constraints appropriately. Write down your assumptions if you need to make any.

   (b) Translate the above E-R diagram to a relational model, in particular, specify your primary key and foreign key constraints clearly. Tha is, give SQL CREATE statements that define the tables you need.

   (c) Write an SQL statement to create a view that gives the project name and group name of completed projects.

Solution:

(a) The E-R diagram is shown below.



(b) The following CREATE SQL statements create the tables and constraints that correspond to the above E-R diagram.

```
CREATE TABLE Student (
   sid   INTEGER,
   sname CHAR(30),
   email CHAR(50),
   gname CHAR(30),
   PRIMARY KEY sid,
   FOREIGN KEY gname REFERENCES Group)

CREATE TABLE Project (
   pname CHAR(100),
   description CHAR(300),
   deadline DATE,
   status CHAR(15),
   PRIMARY KEY pname,
   CHECK(status IN ('Completed', 'Inprogress')))

CREATE TABLE Group (
   gname CHAR(30),
```

```
grade INTEGER,
pname CHAR(100),
PRIMARY KEY gname,
FOREIGN KEY pname REFERENCES  PROJECT)
```

Since the relations between the entities are not of a many-to-many or many-to-one relationship, we do not create any separate tables for them. The pname attribute is added to the GROUP table and the gname attribute to the STUDENT table.

(c) The view of completed projects:

```
CREATE VIEW  COMPLETEDPROJECT AS
SELECT P.pname, G.gname
FROM  GROUP G,  PROJECT P
WHERE G.pname = P.pname
  AND P.status = 'Completed'
```

Problems for Chapter 5: Relational Algebra and SQL

1. Calculate the **Cartesian product** (also known as *cross product*) of the following two table instances.

A	B
a1	b1
a2	b2
a3	b3

C	D
c1	d1
c2	d2

Solution:

A	B	C	D
a1	b1	c1	d1
a1	b1	c2	d2
a2	b2	c1	d1
a2	b2	c2	d2
a3	b3	c1	d1
a3	b3	c2	d2

2. Can there be a commutativity transformation for the projection operator? Why?

   Solution: No. Think for yourself (just follow the definition).

3. Consider the following relational schema, where the keys are underlined:

   EMPLOYEE (ssn, name, gender, address, salary, supervisorSSN, dnumber)
   DEPARTMENT (dnumber, dmname, managerSSN)
   DEPTLOCATION (dnumber, dlocation)
   PROJECT (pnumber, pname, plocation)
   WORKSON (emplSSN, pnumber, hours)

   Answer the following queries using relational algebra:

   (a) Retrieve the names of all male employees in department number 666, who work more than 13 hours per week on project "Catch 22".

   (b) Find the names of employees who are **two** managerial levels below the employee "Joe Public".

   (c) Find names of the department that have no employees.

   (d) Find those employees in department number 123 who do not work on any project at location NYC.

Solution:

(a) $\pi_{\texttt{name}}($ $\sigma_{\texttt{dnumber=666 AND gender='male'}}(\text{EMPLOYEE})$ $\bowtie_{\texttt{ssn=emplSSN}}$
$(\sigma_{\texttt{pname='Catch22'}}(\text{PROJECT}) \bowtie \sigma_{\texttt{hours>13}}(\text{WORKSON}))$)

(b) $\pi_{\texttt{name}}($ $\text{EMPLOYEE} \bowtie_{\texttt{supervisorSSN=ssn}} (\text{EMPLOYEE} \bowtie_{\texttt{supervisorSSN=ssn2}} \sigma_{\texttt{name2='JoePublic'}}$
$(\text{EMPLOYEE}[\texttt{ssn2}, \texttt{name2}, \texttt{gender2}, \texttt{address2}, \texttt{salary2}, \texttt{supervisorSSN2}, \texttt{dnumber2}])$)

(c) $\pi_{\texttt{dmname}}(\text{DEPARTMENT}) - \pi_{\texttt{dmname}}(\text{DEPARTMENT} \bowtie \text{EMPLOYEE})$

(d) $\pi_{\texttt{name}}(\sigma_{\texttt{dnumber=123}}(\text{EMPLOYEE})) - \pi_{\texttt{name}}($ $\sigma_{\texttt{dnumber=123}}(\text{EMPLOYEE})$ $\bowtie_{\texttt{ssn=emplSSN}}$
$(\sigma_{\texttt{plocation='NYC'}}(\text{PROJECT}) \bowtie \sigma(\text{WORKSON}))$)

4. Consider the following schema: MESSAGE(mId, subject, body), CUSTOMER(email, name, householdSize, address), SENTTO(mId, email, sendDate).

   (a) Write an SQL query that returns all attributes of customers that have household size greater than one.

   (b) Write an SQL query that counts the total number of messages.

   (c) Write an SQL query that returns all email addresses and names of customers who have been sent the message with subject "Spring 2004 Specials".

Solution:

(a)
```
SELECT *
FROM Customer
WHERE householdSize > 1
```

(b)
```
SELECT COUNT(*)
FROM  MESSAGE
```

(c)
```
SELECT C.email, C.name
FROM  MESSAGE M,  SENTTO S,  CUSTOMER C
WHERE M.subject = 'Spring 2004 Specials'  AND  M.mId = S.mId
          AND  S.email = C.email
```

5. Answer the following questions:

   (a) Specify the steps needed for evaluating an SQL statement of the form

   ```
   SELECT thing1
   FROM thing2
   WHERE thing3
   GROUP BY thing4
   HAVING thing5
   ORDER BY thing6
   ```

   (b) Identify which steps in the solution to the problem (a) correspond closely to each of the following operators in relational algebra: (i) select, (ii) project, (iii) cross product, (iv) join.

   (c) Design SQL queries that return each of the following expressions, assuming that R and S are tables representing the relations R and S, respectively: (i) R ∪ S, (ii) R ∩ S (iii) R − S, (iv) R / S.

   For (i) to (iii), you may also assume that R and S are union compatible. For (iv), you may assume that S has attributes b1, ..., bm, and R has in addition attributes a1, ..., an.

Solution:

(a) 1. Take all tables listed in thing2, compute their cross product
2. Select rows of the cross product that satisfy the condition in thing3
3. Group the selected rows by attribute thing4
4. Select the groups that satisfy the condition in thing5
5. Project out for the selected groups attributes in thing1
6. Sort the result by attributes in thing6

(b) (i) steps 2 and 4, (ii) step 5, (iii) step 1, (iv) steps 1 and 2 together

(c) i.
```
SELECT * FROM R UNION SELECT * FROM S
```
ii.
```
SELECT * FROM R INTERSECT SELECT * FROM S
```
iii.
```
SELECT * FROM R EXCEPT SELECT * FROM S
```
iv.
```
SELECT a1,...,an
FROM R R1
WHERE NOT EXISTS(
      SELECT * FROM S
      EXCEPT
      SELECT R2.b1,...,R2.bm
      FROM   R R2
      WHERE  R1.a1=R2.a1 AND ... AND R1.an=R2.an)
```

6. Suppose table USER has attributes email, name, address, and householdSize, where email being the primary key.

   (a) Express in relational algebra the query that finds all pairs of users where the two people both claim to have a household size 2 and have the same address and returns their names and the common address.

   (b) Express the above query in SQL.

   (c) Write in SQL the query that finds the users whose household size is at least 50% more than the average household size and returns their name and household size, sorted by household size. (Hint: decompose the problem into subproblems, and you may use a view to capture a intermediate result.)

   (d) Write in SQL the query that finds all users each having a household size different from the total number of users having the same address as him or her. (Hint: again, decompose the problem into subproblems, and you may use a view to capture a intermediate result.)

Solution:

(a) $\pi_{\texttt{name,name2,address}}(\sigma_{\texttt{householdSize=2}}(\text{USER}) \bowtie_{\texttt{email<>email2 AND address=address2}} \sigma_{\texttt{householdSize2=2}}(\text{USER})[\texttt{email2}, \texttt{name2}, \texttt{address2}, \texttt{householdSize2}])$

(b)
```
SELECT U.name, U2.name, U.address
FROM   USER U,  USER U2
WHERE U.name <> U2.name AND U.householdSize=2 AND U2.householdSize=2
        AND U.address=U2.address
```

(c)
```
CREATE VIEW  AVGHSIZE (avghsize) AS
SELECT AVG(U.householdSize)
FROM  USER U

SELECT U.name, U.householdSize
FROM  USER U,  AVGHSIZE A
WHERE U.householdSize >= 1.5 * A.avghsize
ORDERED BY householdSize
```

(d)
```
CREATE VIEW  ADDRANDCOUNT (address,count) AS
SELECT U.address, COUNT(*)
FROM  USER U
GROUP BY U.address

SELECT U.*
FROM   USER U,  ADDRANDCOUNT A
WHERE U.address=A.address AND U.householdSize <> A.count
```

7. (a) Consider the schema PERSON(id,name,age). Write an SQL query that finds the 100th oldest person in the relation. A 100th oldest person is one such that there are 99 people who are strictly older. (There can be several such people who might have the same age, or there can be none).

   (b) Consider the schema GRADES(id,grade), where grades are in between 0 and 100. Write a query that computes the histogram of the form

```
score count
 10     3     //3 people got 100
  9    20     //20 people got between 90-99
  8    40     //40 people got between 80-89
  :     :     //etc
  0     1     //1 person got between 0-9
```

   This problem is not as easy as it sounds.

Solution:

(a)
```
SELECT P.id, P.name
FROM   PERSON P
WHERE  99 = (SELECT COUNT(*) FROM   PERSON PP WHERE  PP.Age > P.Age)
```

(b)
```
CREATE VIEW  SCORE (score) AS
SELECT floor(grade/10)
FROM   GRADES

SELECT score, COUNT(*)
FROM   SCORE
GROUP BY score
ORDER BY DESC score
```

Note: This is only a partial solution, since, say, if 0 people got 20-29, it misses the pair $\langle 2 \quad 0 \rangle$. For a complete solution, we can create a constant table, DUMMYSCORES, with the attribute `score` and the following tuples: `0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10` (each number is a tuple by itself). Then we can proceed as follows:

```
CREATE VIEW  SCORE1 (score) AS
(SELECT floor(grade/10)
 FROM   GRADES)
UNION
(SELECT * FROM  DUMMYSCORES)

SELECT score, COUNT(*)-1
FROM   SCORE1
GROUP BY score
ORDER BY DESC score
```

This solution is correct because DUMMYSCORES adds 1 to the count of every score in the histogram, which we correct by subtracting 1 in the second query.

8. Consider the following relations:

   BANK (BName, City)
   COURIER (CName, City)

   Produce a relational algebra expression to show the names of the couriers located in every city where Chase Bank is located.

   Solution:

   $\pi_{\texttt{CName,City}}(\text{COURIER}) / \pi_{\texttt{City}}(\sigma_{\texttt{BName='Chase'}}(\text{BANK}))$

9. Express the previous query using SQL.

   Solution:

```
SELECT C1.CName
FROM Courier C1
WHERE NOT EXISTS
    (
        (SELECT B.City
         FROM Bank B
         WHERE B.BName='Chase')
         EXCEPT
        (SELECT C2.City
         FROM Courier C2
         WHERE C1.CName = C2.CName)
    )
```

10. Consider a relation schema `Computer(Model, Speed, Price)`

    For each speed of computer above 800MHz, find the average price.

    Solution:

```
SELECT Speed, AVG(Price)
FROM Computer
WHERE Speed > 800
GROUP BY Speed;
```

11. Consider the relation schemas

```
Computer(Model, Manufacturer, Price)
User(Name, ComputerModel)
Employee(Name, WeeklySalary)
```

where Model and Name are keys in their respective relations.

(a) Find all users such that the average price of their computers is greater than 2000 and their speed is at least 1000MHz. Provide a solution that uses no nested queries but does use HAVING.

(b) Find every manufacturer who makes at least one computer whose price is more than the average weekly salary of the users who use computers from that manufacturer.

Solution:

(a)
```
SELECT U.Name
FROM User U, Computer C
WHERE U.ComputerModel = C.Model
GROUP BY U.Name
HAVING AVG(C.Price) > 2000 AND MIN(C.Speed) >= 1000
```

(b) The following query is a good approximation, but is incorrect:

```
SELECT C.Manufacturer
FROM  COMPUTER C
WHERE   C.Price > ( SELECT AVG(E.WeeklySalary)
                    FROM  USER U,  EMPLOYEE E,  COMPUTER R
                    WHERE U.Name = E.Name
                          AND U.ComputerModel = R.Model
                          AND R.Manufacturer = C.Manufacturer
                  )
```

The reason why it is incorrect is rather subtle: the same employee, E, can use different models from the same manufacturer and such an employee might appear in multiple tuples in the join of USER, EMPLOYEE, and COMPUTER. Therefore, the salary of this user will be counted several times and the average will be wrong. Note that if we used the inner query

```
SELECT AVG(E.WeeklySalary)
FROM  USER U,  EMPLOYEE E,  COMPUTER R
WHERE U.Name = E.Name AND U.ComputerModel = C.Model
```

then we would have the aforesaid problem, but the query would still be incorrect, since it would count the average salary of the employees who use a particular model (C.Model) of a computer, not of all the people who use computers from C.Manufacturer.

To get a correct query, we need to use a nested correlated subquery in the FROM clause:

```
SELECT C.Manufacturer
FROM  COMPUTER C
WHERE   C.Price > ( SELECT AVG(P.WeeklySalary)
                    FROM
                        (SELECT DISTINCT E.Name, E.WeeklySalary
                         FROM  USER U,  EMPLOYEE E,  COMPUTER R
                         WHERE U.Name = E.Name
                               AND U.ComputerModel = R.Model
                               AND R.Manufacturer = C.Manufacturer
                        )  AS  P
                  )
```

Here the innermost SELECT clause finds the set of all users for `C.Manufacturer` and makes sure that every user appears exactly once. Then the average weekly salary is computed.

Problems for Chapter 6: Database Design with the Relational Normalization Theory

1. (a) What does the functional dependency X → Y mean?

   (b) How to use functional dependency to determine keys of a given schema?

   Solution:

   (a) If a pair of tuples has the same values in the attributes of X then they have the same values in the attributes of Y.

   (b) A key is functional dependency X→Y such that
   (1) Y contains all attributes of the schema, and
   (2) There is no functional dependency X'→Y for any subset X' of X.

2. Consider a schema S with functional dependencies:
   {A→BC, C→FG, E→HG, G→A}.

   (a) Compute the attribute closure of A with respect to S.

   (b) Suppose we decompose S so that one of the subrelations, say called R, contains attributes AFE only. Find a projection of the functional dependencies in S on R (i.e., find the functional dependencies between attributes AFE).

   (c) Is R in BCNF? Explain. If not, use one cycle of the BCNF decomposition algorithm to obtain two subrelations, and answer whether they are in BCNF.

   (d) Show that your decomposition is lossless.

   (e) Is your decomposition dependency preserving? Explain.

   (f) Is R in 3NF? Explain. If not, use 3NF decomposition algorithm to obtain subrelations, and answer whether they are in 3NF.

Solution:

(a) ABCFG

(b) {A→F, E→AF} or {A→F, E→A, E→ F} or {A→F, E→A}

(c) No, A is not a superkey. R1={AF,A→F} and R2={EA,E→A}. They are in BCNF.

(d) {AF} intersection {EA} is {A}, which is a key of R1.

(e) Yes, because each of the FDs of R is entailed by FDs of R1 and R2.

(f) No, F is not part of a key. Obtain same R1 and R2. They are in 3NF.

3. Prove that the following sets of FDs are equivalent:

Set 1	Set 2
$A \rightarrow B$	$A \rightarrow C$
$C \rightarrow A$	$C \rightarrow B$
$AB \rightarrow C$	$CB \rightarrow A$

Solution: Let us denote the first set $\mathcal{F}$ and the second $\mathcal{G}$.

$\mathcal{F}$ **entails** $\mathcal{G}$**:** $A^+_{\mathcal{F}} = ABC$, so FD1 in $\mathcal{G}$ is entailed by $\mathcal{F}$.

Similarly, $C^+_{\mathcal{F}} = CAB$, so FD2 in $\mathcal{G}$ is also entailed. Finally, FD3 in $\mathcal{G}$ is entailed by FD2 in $\mathcal{F}$.

$\mathcal{G}$ **entails** $\mathcal{F}$**:** $A^+_{\mathcal{G}} = ACB$, so FD1 in $\mathcal{F}$ is entailed by $\mathcal{G}$. $C^+_{\mathcal{G}} = CBA$, so FD2 in $\mathcal{F}$ is also entailed by $\mathcal{G}$. Finally, FD3 in $\mathcal{F}$ is entailed by FD1 in $\mathcal{G}$.

4. Consider the following functional dependencies over the attribute set BCGHMVWY:

   W → V
   WY → BV
   WC → V
   V → B
   BG → M
   BV → Y
   BYH → V
   M → W
   Y → H
   CY → W

   Find a minimal cover, then decompose into lossless 3NF. After that, check if all the resulting relations are in BCNF. If you find a schema that is not, decompose it into a lossless BCNF. Explain all steps.

   Solution:

   Minimal cover: First split WY → BV into WY → B and WY → V. The next step is to reduce the left-hand sides of the FDs. Since W → B and W → V are implied by the given set of FDs, we can replace WY → B, WY → V, and WC → V in the original set with W → B (W → V already exists in the original set, so we discard the duplicate). Likewise V → Y and BY → V can be derived from the original set so we can replace BV → Y and BYH → V with V → Y and BY → V.

   In the next step, we remove redundant FDs, of which we find only W → B. The final result is therefore

   W → V
   V → B
   BG → M
   V → Y
   BY → V
   M → W
   Y → H
   CY → W

   Lossless 3NF: (WV; {W → V}), (VBY; {BY → V, V → Y, V → B}),

   (BGM; {BG → M}), (MW; {M → W}), (YH; {Y → H}), (CYW; {CY → W).

   Since BGM is a superkey of the original schema, we don't need to add anything to this decomposition.

BCNF: The schema (BGM; {BG → M}) is not in BCNF because M → B is entailed by the original set of FDs and (CYW; {CY → W) is not in BCNF because $W \rightarrow Y$ is entailed.

We decompose BGM with respect to M → B into (BM; {M → B}) and (GM; {}), thereby loosing the FD BG → M.

Similarly CYW is decomposed using W → Y into (WY; {W → Y}) and (WC; {}), loosing CY → W.

5. Consider the schema $\mathbf{R}$ over the attributes ABCDEFG with the following functional dependencies:

$$\begin{aligned}&\mathtt{AB} \rightarrow \mathtt{C}\\&\mathtt{C} \rightarrow \mathtt{B}\\&\mathtt{BC} \rightarrow \mathtt{DE}\\&\mathtt{E} \rightarrow \mathtt{FG}\end{aligned}$$

and the following multivalued dependencies:

$$\begin{aligned}\mathbf{R} &= \mathtt{BC} \bowtie \mathtt{ABDEFG}\\\mathbf{R} &= \mathtt{EF} \bowtie \mathtt{FGABCD}\end{aligned}$$

Decompose this schema into 4NF while trying to preserve as many functional dependencies as possible. Hint: first use the 3NF synthesis algorithm, then the BCNF algorithm, and finally the 4NF algorithm.

Will the resulting schema decomposition be dependency-preserving?

Solution:

First split the right-hand sides of the FDs and minimize the left-hand sides. You will get:

$$\begin{aligned}&\mathtt{AB} \rightarrow \mathtt{C}\\&\mathtt{C} \rightarrow \mathtt{B}\\&\mathtt{C} \rightarrow \mathtt{D}\\&\mathtt{C} \rightarrow \mathtt{E}\\&\mathtt{E} \rightarrow \mathtt{F}\\&\mathtt{E} \rightarrow \mathtt{G}\end{aligned}$$

We do not show all steps in the above. As an example, we show why $\mathtt{BC} \rightarrow \mathtt{D}$ can be replaced with $\mathtt{C} \rightarrow \mathtt{D}$, *i.e.*, the left-hand side of the original dependency can be reduced to C. To this end, we need to show that $\mathtt{C} \rightarrow \mathtt{D}$ is entailed by the original set of FDs. Compute $\mathtt{C}^+$ with respect to this original set: CBDEFG. Since this closure contains D, it means that $\mathtt{C} \rightarrow \mathtt{D}$ is entailed and therefore we can replace $\mathtt{BC} \rightarrow \mathtt{D}$ with $\mathtt{C} \rightarrow \mathtt{D}$.

Synthesize the 3NF schemas: $\mathbf{R}_1 = (\mathtt{ABC}, \{\mathtt{AB} \rightarrow \mathtt{C}\})$, $\mathbf{R}_2 = (CBDE, C \rightarrow BDE)$, $\mathbf{R}_3 = (EFG, E \rightarrow FG)$.

$\mathbf{R}_1$ is not in BCNF, because $\mathtt{C} \rightarrow \mathtt{B}$ is implied by by our original set of FDs and all its attributes belong to $\mathbf{R}_1$. Therefore, $\mathtt{C} \rightarrow \mathtt{B}$ must hold in $\mathbf{R}_1$ but C is not a superkey of $\mathbf{R}_1$. So, we decompose $\mathbf{R}_1$ further using $\mathtt{C} \rightarrow \mathtt{B}$: $(\mathtt{AC}, \{\,\})$ and $(\mathtt{BC}, \mathtt{C} \rightarrow \mathtt{B})$.

Now we still have two MVDs left. Note that $\mathbf{R} = \mathtt{BC} \bowtie \mathtt{ABDEFG}$ projects onto $\mathbf{R}_2$ as $\mathbf{R}_2 = \mathtt{BC} \bowtie \mathtt{BDE}$ and it violates 4NF in $\mathbf{R}_2$ because $\mathtt{B} = \mathtt{BC} \cap \mathtt{BDE}$ is not a superkey there. So, we can use this MVD to decompose $\mathbf{R}_2$ into $(\mathtt{BC}, \mathtt{C} \rightarrow \mathtt{B})$ and $(\mathtt{BDE}, \emptyset)$.

Similarly, $\mathbf{R} = \mathtt{EF} \bowtie \mathtt{FGABCD}$ projects onto $\mathbf{R}_3$ as $\mathbf{R_3} = \mathtt{EF} \bowtie \mathtt{FG}$. This makes $\mathbf{R}_3$ violate 4NF, and we decompose it into $(\mathtt{EF}, \mathtt{E} \rightarrow \mathtt{F})$ and $(\mathtt{FG}, \emptyset)$.

The decomposition is not dependency preserving. For instance, the FD $\mathtt{A} \rightarrow \mathtt{B}$, which was present in the original schema, is now not derivable from the FDs that are attached to the schemas in the decomposition.

6. Consider the schema **R** over the attributes $ABCDEFG$ with the following functional dependencies:

$$DE \to F$$
$$BC \to AD$$
$$FD \to G$$
$$F \to DE$$
$$D \to E$$

and the following multivalued dependency:

$$\mathbf{R} = ABCFG \bowtie BCDE$$

Decompose this schema into 4NF using the following method: first obtain a BCNF decomposition using the FDs only. Then further normalize to 4NF using the MVD, if necessary.

Solution:

Minimal cover.

Split the LHS:

1. $DE \to F$
2. $BC \to A$
3. $BC \to D$
4. $FD \to G$
5. $F \to D$
6. $F \to E$
7. $D \to E$

Reduce the RHS 3: Delete E from the LHS of FD 1, since $F \in E^+$. The LHSs of FDs 2,3 cannot be reduced. D can be deleted from the LHS of FD 4 since $G \in F^+$. Result:

1′. $D \to F$
2′. $BC \to A$
3′. $BC \to D$
4′. $F \to G$
5′. $F \to D$
6′. $F \to E$
7′. $D \to E$

Remove redundant FDs: FD 6' (or FD 7') is redundant and can be deleted:

1′. $D \to F$
2′. $BC \to A$
3′. $BC \to D$
4′. $F \to G$
5′. $F \to D$
7′. $D \to E$

Note: one can delete *either* FD6' *or* FD7' — *not* both.

Construct 3NF: $(DEF; D \rightarrow EF)$, $(BCAD; BC \rightarrow AD)$, $(FGD; F \rightarrow GD)$. This decomposition is lossless since $BCAD^+ = BCADEFG$, *i.e.*, the attribute set of the second schema is a superkey of the original schema.

BCNF: Note that $F \rightarrow D$ applies to schema 1 and $D \rightarrow F$ applies to schema 3 (these FDs are entailed by the original set). However, these FDs do not violate BCNF in either of these schemas. So, the schema obtained in the previous step is already in BCNF.

4NF: The only non-trivial projection of the MVD $\mathbf{R} = ABCFG \bowtie BCDE$ on the above schemas is the projection on $BCAD$: $BCAD = ABC \bowtie BCD$. However, $ABC \cap BCD = BC$ is a superkey of the schema $(BCAD; BC \rightarrow AD)$. Therefore, this MVD does not violate the 4NF and no further action is required.

7. Suppose schema **S** has attributes `A`, `B`, and `C`. The attribute `A` uniquely determines `B` and `C` uniquely determines `A`. Suppose we decompose **S** into **S1** and **S2** such that **S1** contains `A` and `B`, and **S2** contains `C` and `A`.

   (a) What are the keys of **S**, **S1**, and **S2**, respectively?

   (b) Does the decomposition remove redundancy? Why?

   (c) Is the decomposition lossless? Why?

   (d) Is the decomposition dependency-preserving? Why?

   (e) Does the decomposition lead to tables that are overall smaller? Why?

   (f) Does the decomposition lead to tables for which it is easier to guarantee the given integrity constraints? Why?

   (g) Does the decomposition lead to tables for which it is easier to write query operations? Why?

   (h) Does the decomposition lead to tables that enable faster query and update operations? Why?

   Solution:

   (a) The keys are `C`, `A`, and `C`, respectively.

   (b) Yes. There is redundancy in **S**, because all rows with same value of `A` have also the same value of `B`. There is no redundancy in **S1** and **S2**.

   (c) Yes. The intersection of **S1** and **S2** is `A`, which is a key of **S1**.

   (d) Yes. The functional dependencies in **S** are equivalent to the union of the dependencies in **S1** and **S2**.

   (e) Yes and No. Yes if many rows in **S** have the same value of `A`, since for each value of `A`, the same value of `B` does not need to repeated that many times. Otherwise no, since `A` has to be duplicated in both **S1** and **S2**.

   (f) Yes. Maintaining the constraint that `A` uniquely determines `B` requires no additional work after decomposition.

   (g) No. For queries that need both attributes `B` and `C` two tables must be joined rather than just using one table.

   (h) Yes and No. Yes, if there are many records and there is much redundancy, and thus the table is much bigger before decomposition and may lead to more page faults and slower query evaluation. No for queries otherwise. Yes also for update operations that involve only attributes `B` and `A`.

8. For the attribute set *R=BCEGVWY*, let the MVDs be:

   $R = WBCY \bowtie YEVG$
   $R = WBCE \bowtie WBYVG$
   $R = WBY \bowtie CYEVG$

   Find a lossless decomposition into 4NF. Is it unique?

   Solution:

   Use MVD #1 to obtain the following decomposition: *WBCY*, *YEVG*. MVD #2 applies to *WBCY* and yields *WBC*, *WBY*. MVD #3 cannot be used to decompose *WBC* because the join attribute, *Y*, is not in this attribute set. It cannot be used to decompose *WBY* or *YEVG* because MVD #3 projects as a trivial dependency in these cases: $WBY = WBY \bowtie Y$ and $YEVG = Y \bowtie YEVG$. Thus, the result is *WBC*, *WBY*, *YEVG*.

   The above decomposition is *not* unique. If we first apply MVD #3 and then #1 then will obtain the following result: *WBY*, *CY*, *YEVG*.

Problems for Chapter 7: Triggers and Active Databases

1. (a) What is the basic structure of a trigger?
   (b) What is the difference between row-level and statement-level trigger granularities?
   (c) What is a trigger precondition in general, and what can it be in SQL?
   (d) List at least three of the issues to consider in trigger handling.

   Solution:

   (a) ON event IF precondition THEN action.
   (b) Row-level: Change of a single row is an event.
   Statement-level: A statement that can change multiple rows is a single event.
   (c) An expression that evaluates to true or false based on database states. Any condition allowed in the WHERE clause of SQL.
   (d) When the precondition is checked. When the action is executed. Whether condition can refer to states both before and after the event occurs. How multiple triggers activated by a single event should be handled. Etc.

2. Consider a brokerage firm database with relations HOLDINGS(AccountId, StockSymbol, Price, Quantity) and BALANCE(AccountId, Balance). Write the triggers for maintaining the correctness of the account balance when stock is bought (a tuple is added to HOLDINGS or Quantity is incremented) or sold (a tuple is deleted from HOLDINGS or Quantity is decremented).

   Write both row level and statement level triggers.

   Solution:

```
CREATE TRIGGER UpdateBalanceRealTime
  AFTER INSERT, DELETE, UPDATE ON Holdings
  REFERENCING NEW AS N
  FOR EACH ROW
    UPDATE Balance
    SET Balance =
      (SELECT SUM(H.Price*H.Quantity)
       FROM Holdings H
       WHERE H.AccountId = N.AccountId )
```

   The above trigger is appropriate for real-time updates of HOLDINGS, so the balances are also updated in real time. If HOLDINGS is updated only periodically (e.g., every day), then a statement level trigger would be more efficient. This trigger can work by erasing the old contents of BALANCE and then recomputing it from scratch:

```
CREATE TRIGGER UpdateBalanceAllAtOnce
  AFTER INSERT, DELETE, UPDATE ON Holdings
  FOR EACH STATEMENT
  BEGIN
    DELETE FROM Balance;    -- Erase
    INSERT INTO Balance
      SELECT DISTINCT H.AccountId, SUM(H.Price*H.Quantity)
      FROM Holdings H
      GROUP BY H.AccountId
  END
```

3. Consider the following schema:

   STUDENT(Student,Status)
   TOOK(Student,Course)
   COURSE(Course,Credits,Type)

   In a STUDENT relation, Status can be 'B' (beginner), 'CPR' (completed program requirements), and 'EG' (eligible to graduate). In a COURSE relation, Type can be 'C' (core course) or 'E' (elective course).

   Write the following row-level triggers which monitors insertions into TOOK:

   (a) When a 'CPR' student completes 130 credits, change the student's status to 'EG'.

   (b) When a beginner student completes all core ('C') courses plus 3 electives ('E'), change the status from 'B' to 'CPR'.

   Hint: The main thing in constructing such triggers is to first figure out the conditions in the WHEN clause. This conditions are similar to complex WHERE clauses. For instance, (b) involves relational division (recall how to do such things with NOT EXISTS).

   Solution:

(a)
```
CREATE TRIGGER EligibleToGraduate
    AFTER INSERT ON Took
    REFERENCING NEW AS N
    FOR EACH ROW
    WHEN (
            130 <= ( SELECT SUM(C.Credits)
                     FROM Took T, Course C, Student S
                     WHERE T.Student = N.Student
                             AND T.Course = C.Course
                             AND T.Student = S.Student
                             AND S.Status = 'CPR' )
          )
    UPDATE Student
    SET Status = 'EG'
    WHERE N.Student = Student
```

(b)

```
CREATE TRIGGER DoneWithProgram
  AFTER INSERT ON Took
  FOR EACH ROW
  REFERENCING NEW AS N
  WHEN (
          EXISTS (  -- Student has status 'B'
               SELECT *
               FROM Student S
               WHERE N.Student = S.Student
                         AND S.Status = 'B'
          )
          AND
          NOT EXISTS (
               ( SELECT C.Course
                 FROM Course C
                 WHERE C.Type = 'C' )
               EXCEPT
               ( SELECT T.Course
                 FROM Took T
                 WHERE T.Student = N.Student)
          )
          AND
          3 <= ( SELECT COUNT(T.Course)
                 FROM Took T, Course C
                 WHERE T.Student = N.Student
                          AND T.Course = C.Course
                          AND C.Type = 'E' )
      )
  UPDATE Student
  SET Status = 'CPR'
  WHERE N.Student = Student
```

Problems for Chapter 8: Using SQL in an Application

1. (a) What are the two kinds of interfaces that application programs can use to talk to database servers?

   (b) Which one of the two kinds above does each of the following interface belong to? (i) embedded SQL, (ii) dynamic SQL, (iii)JDBC, (iv) SQLJ, (v) ODBC.

   (c) What is the advantage of embedded SQL over dynamic SQL and vice versa?

   (d) Explain why constraint checking is usually deferred in transaction processing applications.

   (e) What forms a transaction in embedded SQL?

   (f) What is the basic sequence of steps to do in JDBC (all the basic steps that involve interacting with database, e.g., making connection) if a computation involves doing an SQL query or update?

   Solution:

   (a) Statement-level interface (SLI). Call-level interface (CLI).

   (b) (i) SLI, (ii) SLI, (iii) CLI, (iv) SLI, (v) CLI.

   (c) Embedded SQL can be executed more efficiently. Dynamic SQL is more general and flexible.

   (d) Because in many transactions, the intermediate database states violate the integrity constraints but the final states do not.

   (e) From when the first SQL statement is executed till a COMMIT or ROLLBACK statement that follows.

   (f) Import Java SQL API, load database driver, and get connection;
   Create, prepare, execute SQL query stmt, and store result in result set;
   Process result set (read and/or update);
   Free SQL statement object, and close connection.
   (Also, put JDBC code in try and catch of SQLException.)

2. Give an example of a transaction program that contains a cursor, such that the value returned by one of its FETCH statements depends on whether or not the cursor was defined to be INSENSITIVE. Give a one sentence explanation of your answer.

   Assume this transaction is the only transaction executing. We are not concerned about any effect that a concurrently executing transaction might have

   Solution:

   In the program below, if the cursor is INSENSITIVE, the effect of the DELETE statement will not be seen by the FETCH statement.

```
#define OK  "00000"
#define EndOfScan  "02000"
EXEC SQL BEGIN DECLARE SECTION;
      unsigned long stud_id;
      char grade[1];
      char *crs_code;
      char *semester;
      char SQLSTATE[6];
EXEC SQL END DECLARE SECTION;

EXEC SQL DECLARE GetEnrolled INSENSITIVE CURSOR FOR
      SELECT T.StudId, T.Grade
         FROM Transcript T
         WHERE T.CrsCode = :crs_code
           AND T.Semester =:semester
FOR READ ONLY;

EXEC SQL OPEN GetEnrolled;

EXEC SQL DELETE FROM Transcript
      WHERE CrsCode IN (:crs_code);

EXEC SQL FETCH GetEnrolled INTO :stud_id, :grade;
while (strcmp(SQLSTATE,OK) == 0)  {
          ... process the values in stud_id and grade ...
      EXEC SQL FETCH GetEnrolled INTO :stud_id, :grade;
 }

EXEC SQL CLOSE GetEnrolled;
```

Problems for Chapter 9: Physical Data Organization and Indexing

1. (a) What are the goals in designing efficient data storage methods? List two principles in achieving these goals.

   (b) Suppose a phone book contains 500 pages, and each page can contain up to 500 records. Suppose we want to search for a particular name in the book. Give a worst-case bound on the number of pages that must be looked at to perform a search using each of the following methods: (i) linear search, as if the book were organized as a heap file, (ii) binary search, using the fact that the book is ordered by names, (iii) with an index for the name of the first entry on each page.

   (c) What kind of index does CREATE TABLE generally yield? What kind of index does CREATE INDEX generally create? What is the advantage of the former kind of index over the latter kind?

   (d) Can a file have more than one sparse index? Can a file have more than one clustered index? Can an unclustered index be sparse?

   Solution:

   (a) Minimize latency, and reduce the number of page transfers. Store pages containing related information close together on disk, and keep cache of recently accessed pages in main memory.

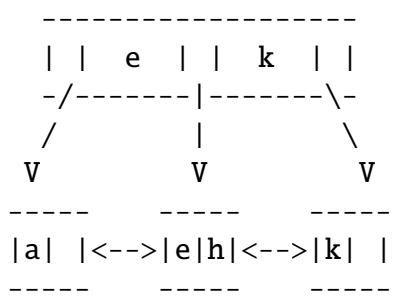
   (b) 500, 9 (=$\lceil log_2 500 \rceil$), 2 (the index has 500 pages, so the entire index fits in one page; so 1 access to the index and 1 to the data).

   (c) Integrated, clustered, main index.
   Separate, unclustered, secondary index.
   Avoids an indirection (1 fewer page), allows to use sparse index, allows efficient range search, supports partial key searches.

   (d) No. No. No.

2. (a) What are the advantages of tree indexing over hash indexing and vice versa?

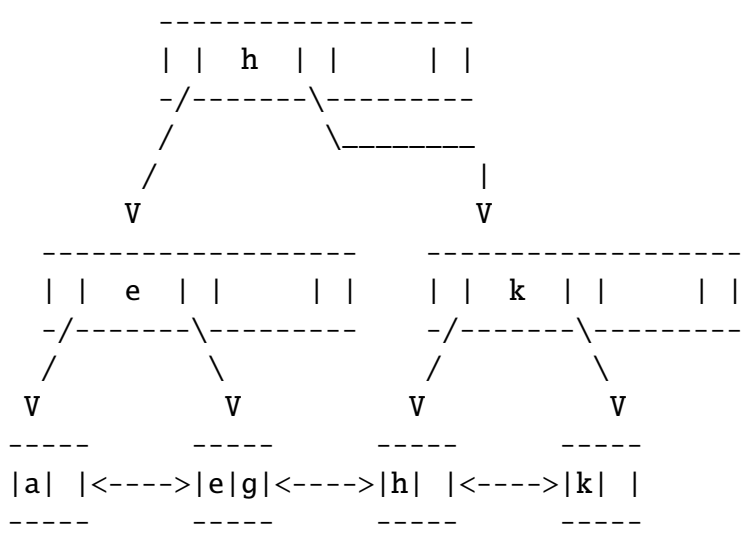
   (b) Show what the following B+ tree looks like after the insertion of g.



Solution:

(a) Supports efficient range search, and partial key search. Is faster if good hash function can be used to eliminate overflow chains.

(b) The root node will be split and the tree will add one level:



3. Consider the following B+-tree, where a node can contain two search key values and three pointers. Suppose that this tree indexes a file where each page can contain 10 records.

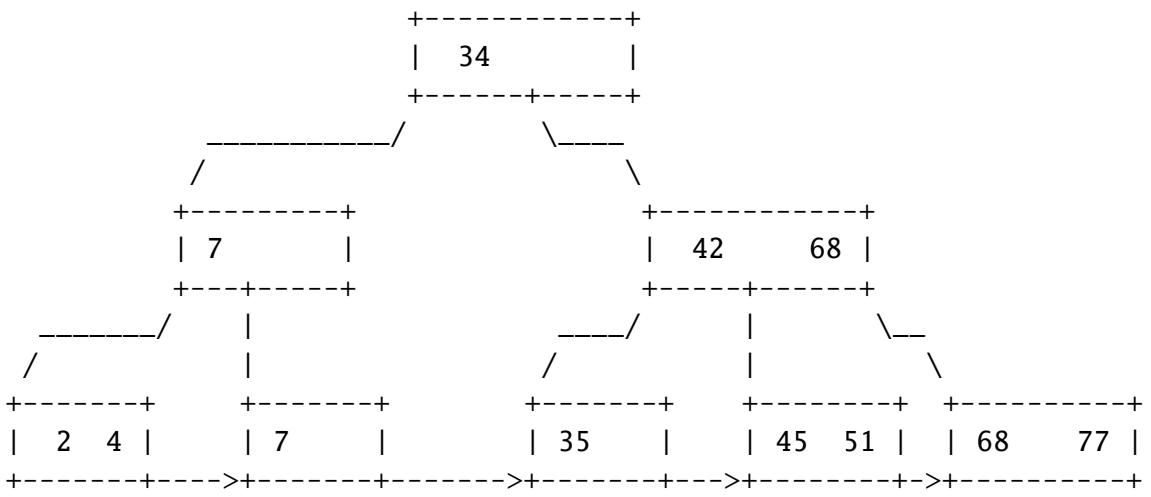
   (a) Assuming that the index is *unclustered*, what is the maximal size (measured in data records) of a file that can be indexed by the depicted B+-tree?

   (b) Same question, but assume that the index is now *clustered*.

   (c) Show the B+-tree after inserting a new record with search key value 70.

   (d) Show the tree after deletion (from the original tree) of the record with search key 7.

   (You must redraw the tree when giving the answer– leave the original tree intact.)

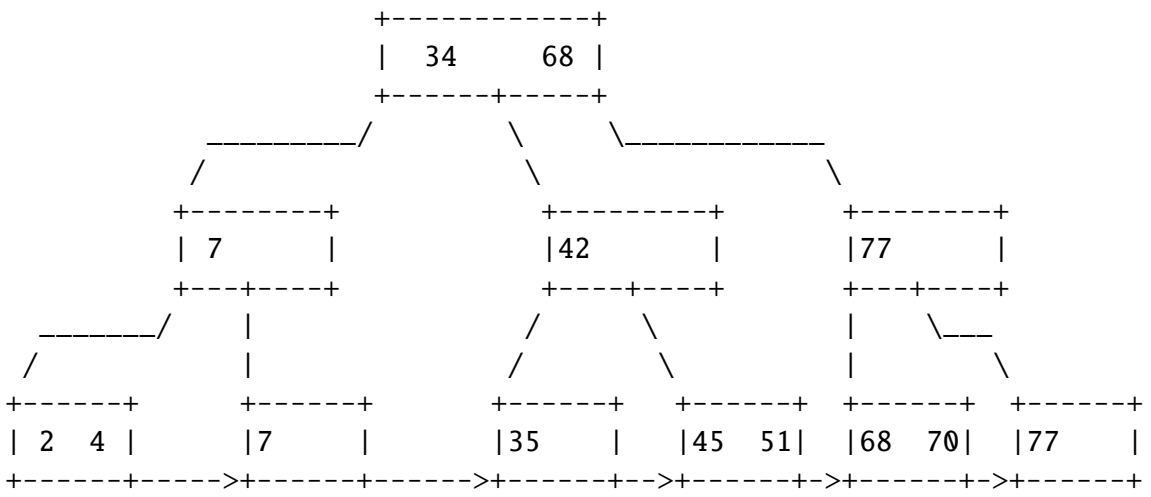


Solution:

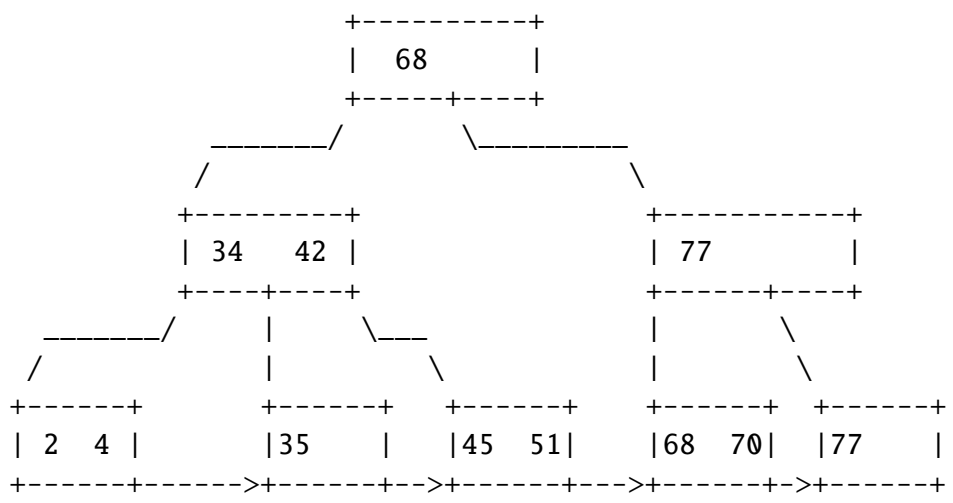
(a) Unclustered index: Each leaf entry can index only a single data record, so the max file size is 8 data records.

(b) Clustered index: the tree can index 8 *pages*, which can contain up to 80 data records.

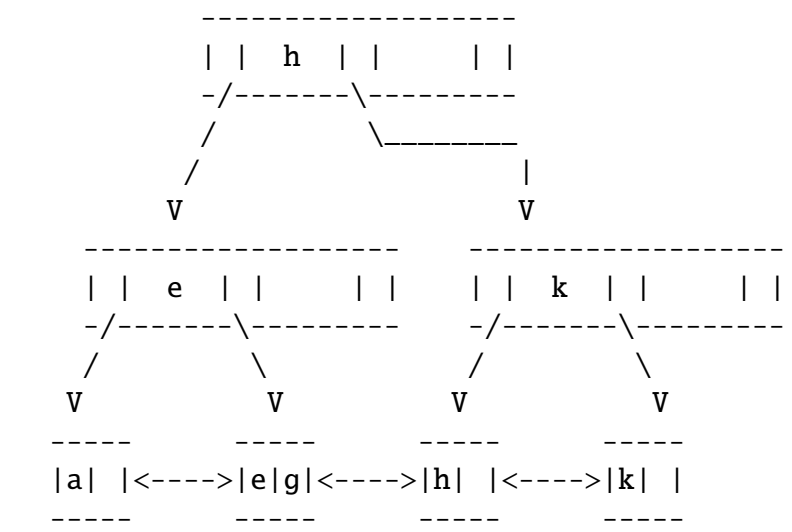
(c) Insertion of 70:



(c) Deletion of 7:



4. Consider the following B+ tree:



(a) How many pages of actual data file does the tree provide index for?

(b) At most how many keys can be inserted in the children of the leftmost node (with the separator key e) in the middle level without splitting that node?

(c) At least how many keys must be inserted so that the left node in the middle level would split?

(d) At most how many keys can be inserted without splitting the root node?

Solution:

(a) 5 pages. Each leaf key in the tree can index 1 page of the data file.

(b) 3 keys. Let us denote the leaf nodes n1, n2, n3, n4 and the mid-level nodes m1, m2. Since we need to find the maximal number of keys that can be inserted in the children of m1 before it gets split, we should try to insert as many as possible keys without affecting any splits.

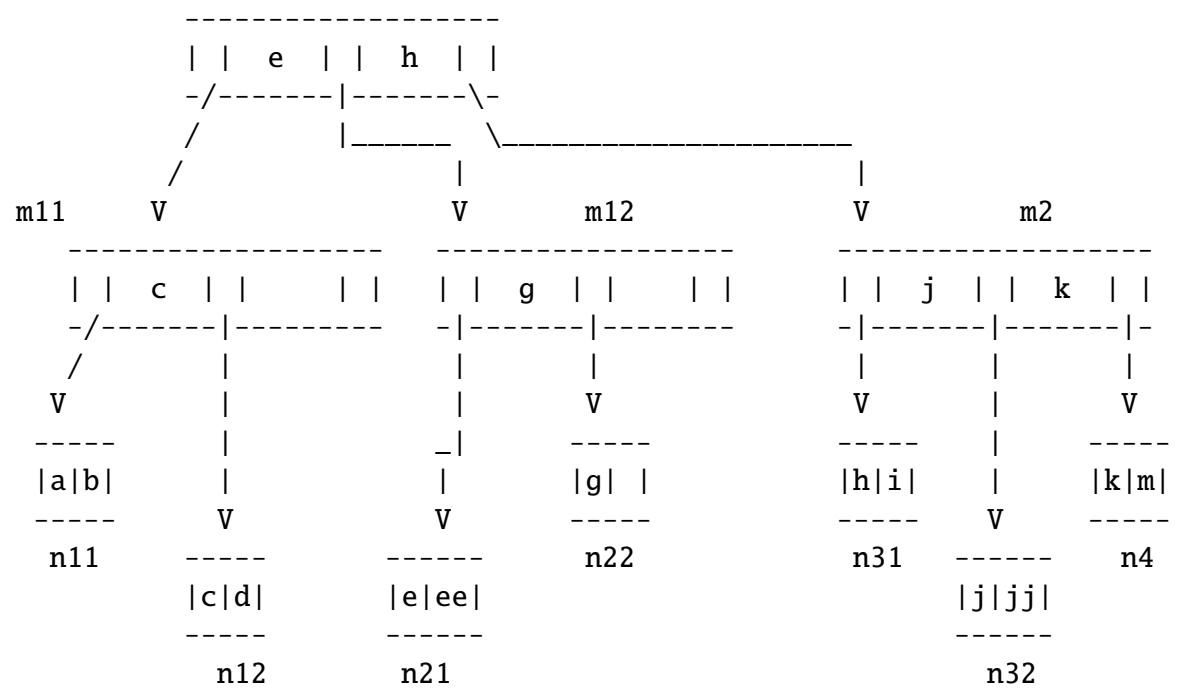
We can insert 1 key into n1. Adding 1 key to n2 splits this node into n21 and n22. A separator key moves up to m1 filling up the remaining free space there. The node n21 is going to be full, but n22 will have one free spot, which we can fill in with one additional key. After that, any addition to the leaves of m1 will necessarily generate a 4th child and thus will cause splitting of m1.

(c) Here we are interested in the *least* number of keys that are required to affect a split in m1. Therefore we should try to add keys in a manner that will cause node splits as quickly as possible.

Adding a key to n2 will cause a split into n21 and n22 with n22 having one free spot and n21 being full. A separator key will be pushed up to m1 filling up the remaining free slot there. Adding a key to n21 will cause a split in that node and will necessitate a push-up of a separator key to m1. This will overflow m1 and cause a split.

(d) 13 keys. This is a tedious, but simple count. As in (a), we should be adding keys as conservatively as possible trying to delay splits as long as possible.

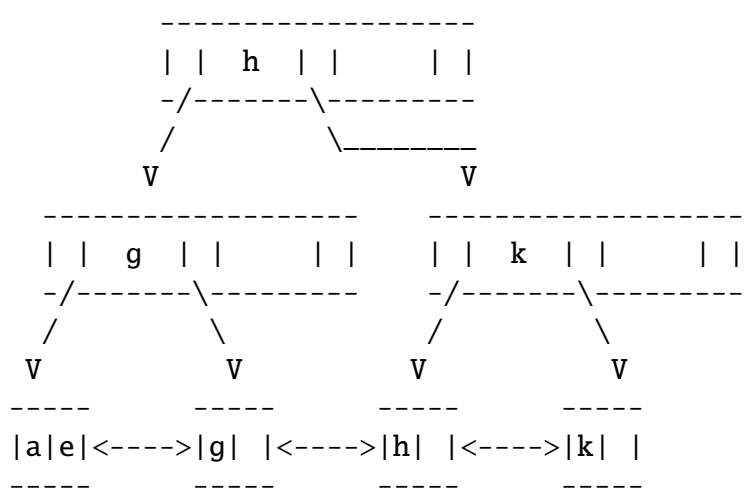
Add 3 keys, b, i, m, to fill up the free spots at the leaves.
Add 2 keys: c to n1 and j to n3. This will split those nodes and will fill up m1 and m2.
Add 2 keys, d and jj, to fill up the free slots in the newly created leaves.
Add 1 key, ee, to n2 to split it into n21 and n22. This will cause a split of m1 into m11 and m12 causing the root to be filled up. This stage of the tree is depicted below:



Now, adding 2 keys to n22 will split it into n221 and n222 and will fill up m12.
Add 1 key to n222 to fill it up.
Add 1 key to n12. This will split it up into n121, n122 and fill up m11. n121 will be full, but n122 will have one free slot.
Add 1 key to n122.

At this stage, all slots in the tree will be occupied and adding any additional key will cause a split at the root. It is easy to see that we have added 3+2+2+1+2+1+1+1=13 keys.

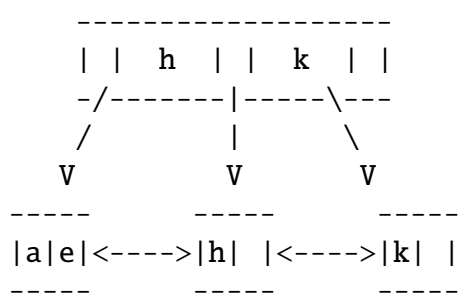
5. Consider the B^+ tree below.



Show what happens after the key g is deleted.

Solution:

In an exam you will need to show all steps. Here is the final result:



After deletion of g from the leaf, the leaf node is deleted as well. The separator key g in the second level is now useless and is deleted. The resulting empty node is merged with its sibling and parent to produce the result.

6. Suppose a family of hash functions h_k(v)=h(v) mod 2^k is used for extendable hashing, and we have the following search key value v and hash function value h(v):

v	bill	jane	karl	mary	tony
h(v)	00010	10111	11001	00010	10101

If currently k=1, the file has two buckets: {bill, mary} and {jane, tony}, and if each bucket can contain at most two records, what happens to k and to the buckets when inserting karl?

Solution:

k=2, and the second bucket is split into two: {jane} and {karl,tony}.

Problems for Chapter 10: The Basics of Query Processing

1. What kind of indexing is usually needed to efficiently evaluate the following query?

```
SELECT E.Id
FROM Employee E
WHERE E.salary <= 100000 AND E.salary >= 30000
```

Solution: Secondary B+ tree index with search key `salary`

2. Consider a relation **s** over the attributes A and B with the following characteristics:

   - 7,000 tuples with 70 tuples per page
   - A hash index on attribute *A*
   - The values that the attribute *A* takes in relation **s** are integers that are uniformly distributed in the range 1 – 200.

   (a) Assuming that the aforesaid index on *A* is *unclustered,* estimate the number of disk accesses needed to compute the query $\sigma_{A=18}(\mathbf{s})$.

   (b) What would be the cost estimate if the index were clustered?

   Explain your reasoning.

   Solution:

   It takes 1.2 I/Os to find the right bucket. The number of tuples selected from **s** is expected to be 7000/200 = 35.

   (a) If the index is unclustered, every tuple can potentially lead to a separate I/O. So, the cost will be ceiling(1.2 + 35) = 37.

   (b) If the index is clustered, then 37 tuples can at most span 2 pages, so the I/O cost would be ceiling(1.2+2)=4.

3. What is the cost of external sorting?

   Solution:

   $2 * F * log_{M-1}F$, where F is the number of pages in file and M is the number of pages in memory.

4. Give an example of an instance of the TRANSCRIPT relation (with the attributes StudId, CrsCode, Semester, and Grade) and a hash function on the attribute sequence ⟨StudId, Grade⟩ that sends two identical tuples in $\pi_{\texttt{StudId,Semester}}(\text{TRANSCRIPT})$ into *different* hash buckets. (This shows that such a hash-based access path cannot be used to compute the projection.)

   Solution: Consider the following two tuples: ⟨111111111, EE101, F1997, A⟩ and ⟨111111111,MAT123,F1997,B⟩. Since these tuples have identical projections on the attributes StudId,Semester, they become one in the projected relation

   $$\pi_{\texttt{StudId,Semester}}(\text{TRANSCRIPT})$$

   Consider now the following hash function on StudId,Grade:

   $$f(t) \;=\; (t.\texttt{StudId} + (t.\texttt{Grade} - \texttt{'A'})) \bmod 1000$$

   The first tuple will then be sent to the bucket 111111111 mod 1000 = 111 and the second to 111111112 mod 1000 = 112.

5. Estimate the cost of **r** $\bowtie$ **s** using

   (a) Sort-merge join

   (b) Block nested loops

   where **r** has 1,000 tuples, 20 tuples per page; **s** has 2,000 tuples, 4 tuples per page; and the main memory buffer for this operation is 22 pages long.

   Solution:

   Relation **r** has 50 pages and **s** has 500 pages. In each case we ignore the cost of outputting the result, as it is the same for both methods.

   (a) Sorting **r** will take 2*50*$log_{21}50 \approx 200$. Sorting **s** will take 2*500*$log_{21}500 \approx 3000$. Thus, assuming that merging can be done during the last phase of sorting, the total cost should not exceed 3200 page transfers.

   (b) Assuming that **r** is scanned in the outer loop, the cost is 50 + (50/20)*500 = 1300.

6. Consider the expression

$$\sigma_{\texttt{StudId=666666666} \wedge \texttt{Semester='F1995'} \wedge \texttt{Grade='A'}}(\textsc{Transcript})$$

Suppose that there are the following access paths:

(a) An unclustered hash index on `StudId`

(b) An unclustered hash index on `Semester`

(c) An unclustered hash index on `Grade`

Which of these access paths has the best selectivity and which has the worst? Compare the selectivity of the worst access path (among the above three) to the selectivity of the file scan.

Solution:

With the unclustered hash index on `StudId`, we will find exactly the bucket that contains all the transcript records for student with the Id 666666666. Since the index is unclustered, this access method will fetch (in the worst case) as many pages as the number of transcript records for that student. In our sample relation in Figure 4.5, this would be 3 pages. In a typical university, an undergraduate student would have to earn 120-150 credits. With 3 credits per course it would make 40-50 transcript records and, thus, the selectivity would be this many pages of data.

With the unclustered hash index on `Semester`, we jump to the bucket for the transcript records in the `F1995` semester and then we need to fetch all these records from the disk to check the other conditions. In a university with enrollment 20,000, selectivity of this access path can be as high as that. In our sample database, however, there are only two transcript records for Fall 1995.

With the unclustered hash index on `Grade`, we get into the bucket of the transcript records where the student received the grade `A`. If only 10% of the students get an `A`, the bucket woud hold 2,000 records per semester. In 20 years (2 semesters a year), the university might accumulate as many as 80,000 transcript records in that bucket. In our sample database, we have 5 transcript records where the student got an `A`.

Thus, in a typical university, the third access path has the worst selectivity and the first has the best. In the sample database of Figure 4.5, the second method has the best selectivity and the third the worst.

7. Compute the cost of $\mathbf{r} \bowtie_{A=B} \mathbf{s}$ using the following methods:

   (a) Nested loops

   (b) Block-nested loops

   (c) Index-nested loops with a hash index on B in **s**. (Do the computation for both clustered and unclustered index.)

   where **r** occupies 2,000 pages, 20 tuples per page, **s** occupies 5,000 pages, 5 tuples per page, and the amount of main memory available for block-nested loops join is 402 pages. Assume that at most 5 tuples in **s** match each tuple in **r**.

   Solution:

   (a) Nested loops: scan **r** and for each of its 40,000 tuples scan **s** once. The result is

   $$2,000 + 40,000 \times 5,000 = 200,002,000 \text{ pages}$$

   (b) Block-nested loops: Scan **s** once per each 400-page block of **r**, i.e., 5 times. The result therefore is:

   $$2,000 + 5,000\lceil \frac{2,000}{402-2} \rceil = 27,000 \text{ pages}$$

   (c) Index-nested loops: The result depends on whether the index on B in **s** is clustered or not. For the clustered case, all tuples of **s** that match a tuple of **r** are in the same disk block and require 1 page transfer (since we assumed that at most 5 tuples of **s** match, they all fit in one disk block). We also need to search the index once per each tuple of **r**. Suppose the later takes 1 disk access. Thus, the total is

   $$2,000 + (1 + 1 * 1.2) \times 40,000 = 90,000 \text{ pages}$$

   In case of an unclustered index, the matching tuples of **s** can be in different blocks. As before, assume that **s** has at most 5 matching tuples per tuple in **r**. Thus, the cost would be

   $$2,000 + (1 + 5 * 1.2) \times 40,000 = 282,000 \text{ pages}$$

Problems for Chapter 11: An Overview of Query Optimization

1. Suppose a database has the following schema:
   TRIP(fromAddrId: INTEGER, toAddrId: INTEGER, date: DATE)
   ADDRESS(id: INTEGER, street: STRING, townState: STRING)

   (a) Write an SQL query that returns the street of all addresses in 'Stony Brook NY' that are destination of a trip on '5/14/02'.

   (b) Translate the SQL query in (a) into the corresponding "naive" relational algebra expression.

   (c) Translate the relational algebra expression in (b) into an equivalent expression using pushing of selections and projections.

   (d) Translate the relational algebra expression in (c) into a most directly corresponding SQL query.

   Solution:

   (a)
   ```
   SELECT A.street
   FROM ADDRESS A, TRIP T
   WHERE A.id=T.toAddrId AND A.townState='Stony Brook NY'
         AND T.date='05/14/02'
   ```

   (b) $\pi_{\texttt{street}}\sigma_{\texttt{id=toAddrId AND townState='StonyBrookNY' AND date='05/14/02'}}(\texttt{Address} \times \text{TRIP})$

   (c) $\pi_{\texttt{street}}(\sigma_{\texttt{townState='StonyBrookNY'}}(\text{ADDRESS}) \bowtie_{\texttt{id=toAddrId}} \sigma_{\texttt{date='05/14/02'}}(\text{TRIP}))$

   (d)
   ```
   SELECT A.street
   FROM (SELECT * FROM ADDRESS WHERE townState='Stony Brook NY') A,
        (SELECT * FROM TRIP WHERE date='05/14/02') T
   WHERE A.id=T.toAddrId
   ```

2. Consider a relation **r** over the attributes A, B, C with the following characteristics:

   - 5,000 tuples with 5 tuples per page
   - Attribute A is a candidate key
   - Unclustered hash index on attribute A
   - Clustered B^+ tree index on attribute *B*
   - Attribute *B* has 1,000 distinct values in **r**
   - Attribute *C* has 500 distinct tuples and an unclustered 3-level B^+ tree index

   (a) Estimate the cost of computing $\sigma_{A=const}(\mathbf{r})$ using the index

   (b) Estimate the cost of computing $\sigma_{B=const}(\mathbf{r})$ using the index

   (c) Estimate the cost of computing $\sigma_{C=const}(\mathbf{r})$ using the index

   (d) Estimate the cost of computing $\pi_{AC}(\mathbf{r})$

   Solution:

   (a) Since the hash index is on the candidate key, $\sigma_{A=const}(\mathbf{r})$ has at most one tuple. Therefore, the cost is 1.2 (searching the index) + 1(retrieving data). If the index is integrated then the cost is just 1.2.

   Note that the fact that even though the hash index is unclustered, we are not paying the price, because the index is on a candidate key.

   (b) Since B has 1,000 distinct values in **r**, there are about 5 tuples per value. Therefore $\sigma_{B=const}(\mathbf{r})$ is likely to retrieve 5 tuples. Because the index is clustered and because there are 5 tuples per page, the result fits in 1 page.

   Therefore, the cost is depth of the tree + 1.

   (c) Since C has 500 values, the selection is likely to produce 10 tuples (5000/50). Pointers to these tuples will be in the same or adjacent leaf pages of the B^+ tree. We conservatively estimate that these tuples will occupy 2 leaves (index entries are typically much smaller than the data file records. Thus, the cost of retrieving all the pointers is 3 (to search the B^+ tree for the first page of pointers in the index) + 1 (to retrieve the second page of the index) = 4.

   Since the index is unclustered, each of the 10 tuples in the result can be in a separate page of the data file and its retrieval may require a separate I/O. Thus, the cost is 4+10 = 14.

   (d) Since we do not project out the candidate key, the projection will have the same number of tuples as the original. In particular, there will be no duplicates and no sorting will be required.

   The output will be about 2/3 of the original size assuming that all attributes contribute equally to the tuple size. Since the original file has 5,000/5=1000 blocks, the cost of the operation is 1,000(scan of the file) + 2/3*1,000 (cost of writing out the result).

3. Write down the sequence of steps needed to transform $\pi_A((\mathbf{R} \bowtie_{B=C} \mathbf{S}) \bowtie_{D=E} \mathbf{T})$ into $\pi_A((\pi_E(\mathbf{T}) \bowtie_{E=D} \pi_{ACD}(\mathbf{S})) \bowtie_{C=B} \mathbf{R})$. List the attributes that each of the schemas **R**, **S**, and **T** *must* have and the attributes that each (or some) of these schemas must *not* have in order for the above transformation to be correct.

   Solution:

   - **R** must have: B (because of the join)
   - **S** must have: ACD (because of π_{ACD})
   - **T** must have: E (because of π_E)
   - These schemas should not have identically named attributes, because otherwise it will not be clear which of the two identically named attributes will be renamed in the joins. In particular, **T** should not have A and C, because $\pi_{ACD}(\mathbf{S})$ clearly suggests that it is expected that **S** will have the attribute A that will survive for the outermost projection π_A to make sense, and the attribute C, which should survive in order for the join with **R** to make sense.

   (a) Associativity of the join: $\pi_A(\mathbf{R} \bowtie_{B=C} (\mathbf{S} \bowtie_{D=E} \mathbf{T}))$

   (b) Commutativity of the join: $\pi_A((\mathbf{S} \bowtie_{D=E} \mathbf{T}) \bowtie_{C=B} \mathbf{R})$

   (c) Commutativity of the join: $\pi_A((\mathbf{T} \bowtie_{E=D} \mathbf{S}) \bowtie_{C=B} \mathbf{R})$

   (d) Pushing projection π_A to the first operand of the join: $\pi_A(\pi_{AC}((\mathbf{T} \bowtie_{E=D} \mathbf{S})) \bowtie_{C=B} \mathbf{R})$

   (e) Pushing projection to the first operand in the innermost join: $\pi_A(\pi_{AC}((\pi_E(\mathbf{T}) \bowtie_{E=D} \mathbf{S})) \bowtie_{C=B} \mathbf{R})$. This is possible if AC are the attributes of **S** and not of **T**.

   (f) Pushing projection to the second operand in the innermost join: $\pi_A(\pi_{AC}((\pi_E(\mathbf{T}) \bowtie_{E=D} \pi_{ACD}(\mathbf{S}))) \bowtie_{C=B} \mathbf{R})$.

   (g) Reverse of pushing a projection: $\pi_A((\pi_E(\mathbf{T}) \bowtie_{E=D} \pi_{ACD}(\mathbf{S})) \bowtie_{C=B} \mathbf{R})$

4. Consider the following relations that represent part of a real estate database:

   AGENT(Id, AgentName)
   HOUSE(Address, OwnerId, AgentId)
   AMENITY(Address, Feature)

   The AGENT relation keeps information on real estate agents, the HOUSE relation has information on who is selling the house and the agent involved, and the AMENITY relation provides information on the features of each house. Each relation has its keys underlined.

   Consider the following query:

   ```
   SELECT  H.OwnerId, A.AgentName
   FROM    House H, Agent A, Amenity Y
   WHERE   H.Address=Y.Address AND A.Id = H.AgentId
                   AND Y.Feature = '5BR' AND H.AgentId = '007'
   ```

   Assume that the buffer space available for this query has 5 pages and that the following statistics and indices are available:

   - AMENITY:

     10,000 records on 1,000 houses, 5 records/page
     Clustered 2-level B^+ tree index on `Address`
     Unclustered hash index on `Feature`, 50 features

   - AGENT:

     200 agents with 10 tuples/page
     Unclustered hash index on `Id`

   - HOUSE:

     1,000 houses with 4 records/page
     Unclustered hash index on `AgentId`
     Clustered 2-level B^+ tree index on `Address`

   Answer the following questions (and explain how you arrived at your solutions).

   (a) Draw a fully pushed query tree corresponding to the above query.

   Solution:

   See Figure 1.

   (b) Find the best query plan to evaluate the above query and estimate its cost.

   Solution:

   We could join HOUSE with AGENT or AMENITY, but in any case it is clear that we should first select HOUSE on `AgentId`, because of the large reduction in size: There are 200 agents, 1000 houses, so agent 007 must be handling about 5 houses. At 4 records per page, this would occupy 2 pages.

   Because the index on `AgentId` is unclustered, it would take 1.2 I/Os to find the bucket and 5 I/Os to fetch the relevant pages: 6.2 I/Os in total.

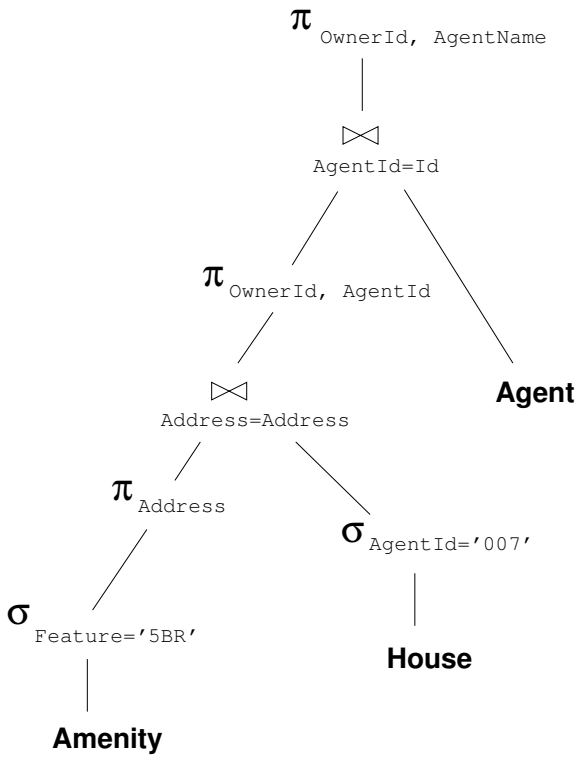


Figure 1:

Next we can join with AGENT, which would take 1.2 page I/Os to search the index, since AGENT has an unclustered index on `Id`, plus 1 I/O to fetch the page — 2.2 I/Os in total. This will still result in 5 tuples, but the tuples will be about 50% larger (AGENT has 10 tuples/page, while HOUSE only 4). However, we can project out `Id` and `AgentId`, which will bring the tuple size to about the size of the tuples in HOUSE. So, the join will still occupy a little over a page. We keep the result of the join in the main memory buffer.

Finally, we join the result with AMENITY. Since the statistics indicate that there are about 10 amenities per house, it doesn't make much sense to select AMENITY on `Feature`: the size will go down by the factor of 10 at a very high price (unclustered hash or scan of 2,000 blocks of the AMENITY relation), and we will loose the index on `Address` — the attribute used in the join.

So, the best way to do the join is to use index-nested loops join using the clustered index on the attribute `Address` of AMENITY. It would take 2 I/Os to search the B^+ tree index for each of the 5 tuples in the result of the previous join (i.e., 2*5; if we cache the top level of the B^+ tree then this search would take 2+4=6 I/Os). The number of tuples retrieved would be 50 (10 features per house * 5 houses), which occupies 10 pages. Therefore, the total needed to retrieve the matching tuples in AMENITY is 16 I/Os.

Note that we still have enough room in the buffer. The expected size of the join is 50 tuples (5*10), which is too large for our 5 page buffer. However, we can also select on `Feature='5BR'` on the fly, reducing the size to about 5 tuples, each about twice the size of the tuples in HOUSE. We can also project)on the fly) on `OwnerId` and `AgentName` further reducing the size. Thus, we will need 2 pages in the buffer for the result of the

join of HOUSE and AGENT, one page for the input buffer that is needed for reading the matching tuples of AMENITY, and two to store the final result. This fits in the available 5 buffer pages.

In total, thus, the query will take 6.2 + 2.2 + 16 = 24.4 I/Os.

(c) Find the next-best plan and estimate its cost.

Solution:

Similar, except that what is left of HOUSE is first joined with AMENITY. The number of I/Os is going to be the same, so this is also a best plan.

The next plan after that would be to do something silly, like joining HOUSE and AGENT using nested loops. Since AGENT occupies 20 blocks, this can be done in 20 I/Os. Then we could proceed to join with AMENITY.

Problems for Chapter 12: Database Tuning

1. Consider the following relational schema:

   STUDENT(Id,Name,Major)
   TOOK(StudId,Course)

   where Id is the primary key in STUDENT. Consider the query

   ```
   SELECT *
   FROM STUDENT S, TOOK T
   WHERE S.Id = T.StudId AND T.Course = 'CS305' AND S.Major = 'EE'
   ```

   (a) Write a relational algebra expression that is equivalent to this query.

   (b) Suggest the indexes that can be added to this database in order to improve the performance of this query. Indicate whether these indices should be clustered or not. Explain your reasoning briefly.

   Solution:

   (a) $\sigma_{\texttt{Course='CS305'}}(\texttt{Took}) \bowtie_{\texttt{StudId=Id}} \sigma_{\texttt{Major='EE'}}(\texttt{Student})$

   (b) Clustered index on Course in TOOK, since it will facilitate the first selection. A clustered index on Major in STUDENT might also help, since it can facilitate the second selection. Note that although Id is a primary key in STUDENT, it might have an unclustered (rather than clustered) index, because numeric single-attribute keys often have unclustered hash indices.

   We could do away with the clustered index on Major, since we could also do the join using the index on Id, which exists because Id is the primary key.

Problems for Chapter 13: An Overview of Transaction Processing

1. Give examples of schedules that have the following properties

   (a) The schedule is serializable, but not in commit order

   Solution:

   $r_1(x)\ w_2(x)\ Commit_2\ r_1(y)\ Commit_1$

   (b) The schedule is serializable, but has a dirty read

   Solution:

   $w_1(x)\ r_2(x)\ Commit_1\ Commit_2$

   (c) The schedule is not serializable, but does not have a dirty read, a non-repeatable read, or a lost update.

   Solution:1 $r_1(x)\ w_2(x)\ r_2(y)\ w_2(y)\ Commit_1\ Commit_2$

   (d) The schedule would be permitted at the READ COMMITTED isolation level, but is serializable.

   Solution:

   $r_1(x)\ r_2(x)\ w_1(x)\ w_2(y)\ Commit_1\ Commit_2$

   (e) The schedule is permitted at READ COMMITTED, but is not serializable

   SOLUTION

   $r_1(x)\ r_2(x)\ w_1(x)\ Commit_1\ w_2(x) Commit_2$

   (f) The schedule is serializable and would be permitted at SNAPSHOT isolation, but not by a strict two-phase locking concurrency control

   Solution:

   $r_1(x)\ r_2(x)\ w_1(x)\ Commit_1\ Commit_2$

   (g) The schedule is serializable and would be permitted by a strict two-phase locking concurrency control, but not at SNAPSHOT isolation

   Solution:

   $r_1(x)\ r_2(y)\ w_2(y)\ Commit_2\ w_1(y)\ \ Commit_1$

2. Explain why

   (a) Obtaining indexes on a table can increase concurrency, as well as decreas the time needed to access the table.

   Solution:

   If index locking is used and the table is accessed through the index, the entire table need not be locked but only the index (in addition to obtaining an intention lock on the table).

   (b) When updating or inserting into a table, increased concurrency is obtained when granular locks are used, even when there are no indices.

   Solution:

   The only locks that need to be obtained are a SIX lock on the table (a read lock and an intention exclusive lock on the table) and a write lock on the rows inserted or updated.

   (c) When using a log to obtain atomicity, the update record must be appended to the log before the database is updated (why it must be a write-ahead log).

   Solution:

   Otherwise if the database is updated and then if the system crashed before the update record was appended to the log, the system would be unable to roll back the database update.

   (d) Asynchronous update systems for updating replicated database might result in inconsistent databases.

   Solution:

   The resulting schedule might not be serializable because, for example, a transaction, T_1 might read two data items, x and y, of a committed transaction T_2, but it might read the value of x that T_2 updated while it was executing and the value of y, which was a replicated value, before it was updated by the transaction that started after T_2 committed.

3. Assume a cohort in the two-phase commit protocol crashes in each of the following situations. When it restarts later, explain what it does and why.

   (a) Before receiving the **prepare** message.

   Solution:

   It aborts because the coordinator will abort the entire transaction when it does not receive a reply to its **prepare** message.

   (b) After receiving the **prepare** message but before it votes

   Solution:

   It aborts because the coordinator will abort the entire transaction when it does not receive a reply to its **prepare** message.

   (c) After sending its **vote** message saying it is ready to commit

   Solution:

   It sends a message to the coordinator asking what the final status of the transaction was and then acts accordingly

   (d) After receiving the **commit** message but before it completes its commit processing

   Solution:

   It completes its commit processing.

Problems for Chapter 16: Introduction to Object Databases

1. Use the following partially defined schema to answer the queries below.

```
CREATE TABLE Student AS
  Id INTEGER,
  Name CHAR(20),
  ...
  Transcript TranscriptType MULTISET

CREATE TYPE TranscriptType
  Course REF(CourseType) SCOPE Course,
  ...
  Grade CHAR(2)
```

The type CourseType is defined as usual, with character string attributes such as `CrsCode`, `DeptId`, etc.

(a) Find all students who have taken more than five classes in the mathematics department.

Solution:

```
SELECT S.Name
FROM Student S
WHERE 5 < ( SELECT count(S1.Transcript->Course)
            FROM   Student S1
            WHERE  S1.Transcript->Course.DeptId = 'MAT'
                   AND  S1.Id = S.Id)
```

(b) Represent grades as a UDT, Grade, with a method, `value()`, that returns the grade's numeric value.

Solution:

```
CREATE TYPE GradeType AS (
   LetterValue CHAR(2) )
METHOD value() RETURNS DECIMAL(3);

CREATE METHOD value() FOR GradeType
RETURNS DECIMAL(3)
LANGUAGE SQL
   BEGIN
      IF LetterValue = 'A' THEN RETURN 4;
      IF LetterValue = 'A-' THEN RETURN 3.66;
      ... ... ...
   END
```

(c) Write a method that, for each student, computes the average grade. This method requires the `value()` method that you constructed for the previous problem.

Solution:

```
SELECT S.Name, avg(S.Transcript.Grade.value())
FROM   STUDENT S
```

Problems for Chapter 17: Introduction to XML and Web Data

1. Specify a DTD appropriate for a document that contains data from both the COURSE table in Figure 4.34 and the REQUIRES table in Figure 4.35. Try to reflect as many constraints as the DTDs allow. Give an example of a document that conforms to your DTD.

   Solution:

   One good example of such a document is shown below. Note that it does not try to mimic the relational representation, but instead courses are modeled using the object-oriented approach.

```
<Courses>
   <Course CsrCode="CS315" DeptId="CS"
           CrsName="Transaction Processing" CreditHours="3">
      <Prerequisite CrsCode="CS305" EnforcedSince="2000/08/01"/>
      <Prerequisite CrsCode="CS219" EnforcedSince="2001/01/01"/>
   </Course>
   <Course>
      .....
   </Course>
   .....
</Courses>
```

   An appropriate DTD would be:

```
<!DOCTYPE Courses [
   <!ELEMENT Courses (Course*)>
   <!ELEMENT Course (Prerequisite*)>
   <!ATTLIST Course
               CrsCode ID #REQUIRED
               DeptId  CDATA #IMPLIED
               CrsName CDATA #REQUIRED
               CreditHours CDATA #REQUIRED >
   <!ATTLIST Prerequisite
               CrsCode IDREF #REQUIRED
               EnforcedSince CDATA #REQUIRED>
]>
```

2. Define the following simple types:

   (a) A type whose domain consists of lists of strings, where each list consists of 7 elements.
   Solution:

```
<simpleType name="ListsOfStrings">
   <list itemType="string" />
</simpleType>
<simpleType name="ListsOfLength7">
   <restriction base="ListsOfStrings">
        <length value="7"/>
   </restriction>
</simpleType>
```

   (b) A type whose domain consists of lists of strings, where each string is of length 7.
   Solution:

```
<simpleType name="StringsOfLength7">
   <restriction base="string">
      <length value="7"/>
   </restriction>
</simpleType>
<simpleType name="ListsOfStringsOfLength7">
   <list itemType="StringsOfLength7" />
</simpleType>
```

   (c) A type appropriate for the letter grades that students receive on completion of a course—A, A−, B+, B, B−, C+, C, C−, D, and F. Express this type in two different ways: as an enumeration and using the `pattern` tag of XML Schema.
   Solution:

```
<simpleType name="gradesAsEnum">
   <restriction base="string">
      <enumeration value="A"/>
      <enumeration value="A-"/>
      <enumeration value="B+"/>
      ..........
   </restriction>
</simpleType>
<simpleType name="gradesAsPattern">
   <restriction base="string">
      <pattern value="(A-?|[BC][+-]?|[DF])"/>
   </restriction>
</simpleType>
```

   In the `gradesAsPattern` representation, (...) represent complex alternatives of patterns separated by | (W3C has adopted the syntax of regular expressions used in Perl), [...] represent simple alternatives (of characters), and ? represents zero or one occurrence, as usual in regular expressions.

3. Write an XML schema specification for a simple document that lists stock brokers with the accounts that they handle and a separate list of the client accounts. The Information about the accounts includes the account `Id`, ownership information, and the account positions (*i.e.*, stocks held in that account). To simplify the matters, it suffices, for each account position, to list the stock symbol and quantity. Use `ID`, `IDREF`, and `IDREFS` to specify referential integrity.

   Solution:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:brk="http://somebrokerage.com/documents"
        targetNamespace="http://somebrokerage.com/documents">
  <element name="Brokerage">
    <complexType>
      <sequence>
        <element name="Broker" type="brk:brokerType"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element name="Account" type="brk:accountType"
                 minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
    </complexType>
  </element>
  <complexType name="brokerType">
    <attribute name="Id" type="ID" />
    <attribute name="Name" type="string" />
    <attribute name="Accounts" type="IDREFS" />
  </complexType>
  <complexType name="accountType">
    <attribute name="Id" type="ID" />
    <attribute name="Owner" type="string" />
    <element name="Positions" />
      <complexType>
        <sequence>
          <element name="Position">
            <complexType>
              <attribute name="stockSym" type="string" />
              <attribute name="qty" type="integer" />
            </complexType>
          </element>
        </sequence>
      </complexType>
    </element>
  </complexType>
</schema>
```

4. Formulate the following XPath queries for the document of the form

```
<Classes>
  <Class CrsCode="CS308" Semester="F1997">
    <CrsName>Software Engineering</CrsName>
    <Instructor>Adrian Jones</Instructor>
  </Class>
  <Class CrsCode="EE101" Semester="F1995">
    <CrsName>Electronic Circuits</CrsName>
    <Instructor>David Jones</Instructor>
  </Class>
  ....
</Classes>
```

   (a) Find the names of all courses taught by Mary Doe in fall 1995.

   Solution:

   ```
   //CrsName[../Instructor="Mary Doe" and ../@Semester="F1995"]/text()
   ```

   Note that we use `text()` at the end to get the text (course names themselves) as opposed to `CrsCode` element nodes.

   (b) Find the set of all document nodes that correspond to the course names taught in fall 1996 or all instructors who taught `MAT123`.

   Solution:

   ```
   //CrsName[../@Semester="F1996"] | //Instructor[../@CrsCode="MAT123"]
   ```

   (c) Find the set of all course codes taught by John Smyth in spring 1997.

   Solution:

   ```
   //Class[Instructor="John Smyth" and @Semester="S1997"]/@CrsCode
   ```