

**Database Systems: An Application-Oriented Approach**  
**(Complete Version)**

Practice Problems and Solutions for Students

Michael Kifer, Arthur Bernstein, Philip M. Lewis

# Contents

|  |            |
|--|------------|
| <b>Problems for Chapter 3: The Relational Data Model</b>   | <b>4</b>   |
| <b>Problems for Chapter 4: Conceptual Modeling of Databases with E-R and UML</b>                     | <b>9</b>   |
| <b>Problems for Chapter 5: Relational Algebra and SQL</b>  | <b>15</b>  |
| <b>Problems for Chapter 6: Database Design with the Relational Normalization Theory</b>              | <b>27</b>  |
| <b>Problems for Chapter 7: Triggers and Active Databases</b>   | <b>37</b>  |
| <b>Problems for Chapter 8: Using SQL in an Application</b>   | <b>41</b>  |
| <b>Problems for Chapter 9: Physical Data Organization and Indexing</b>                               | <b>43</b>  |
| <b>Problems for Chapter 10: The Basics of Query Processing</b>                                       | <b>51</b>  |
| <b>Problems for Chapter 11: An Overview of Query Optimization</b>                                    | <b>58</b>  |
| <b>Problems for Chapter 12: Database Tuning</b>  | <b>66</b>  |
| <b>Problems for Chapter 13: Relational Calculus, Visual Query Languages, and Deductive Databases</b> | <b>71</b>  |
| <b>Problems for Chapter 14: Object Databases</b>   | <b>75</b>  |
| <b>Problems for Chapter 15: XML and Web Data</b>   | <b>80</b>  |
| <b>Problems for Chapter 16: Distributed Databases</b>  | <b>94</b>  |
| <b>Problems for Chapter 17: OLAP and Data Mining</b>   | <b>98</b>  |
| <b>Problems for Chapter 18: Acid Properties of Transactions.</b>                                     | <b>104</b> |
| <b>Problems for Chapter 19: Models of Transactions</b>   | <b>108</b> |
| <b>Problems for Chapter 20: Implementing Isolation</b>   | <b>112</b> |
| <b>Problems for Chapter 21: Isolation in Relational Databases</b>                                    | <b>121</b> |
| <b>Problems for Chapter 22: Atomicity and Durability</b>   | <b>129</b> |
| <b>Problems for Chapter 23: Architecture of Transaction Processing Systems</b>                       | <b>134</b> |
| <b>Problems for Chapter 24: Implementing Distributed Transactions</b>                                | <b>136</b> |
| <b>Problems for Chapter 25: Web Services</b>   | <b>140</b> |
| <b>Problems for Chapter 26: Security and Electronic Commerce</b>                                     | <b>146</b> |



## 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))
```

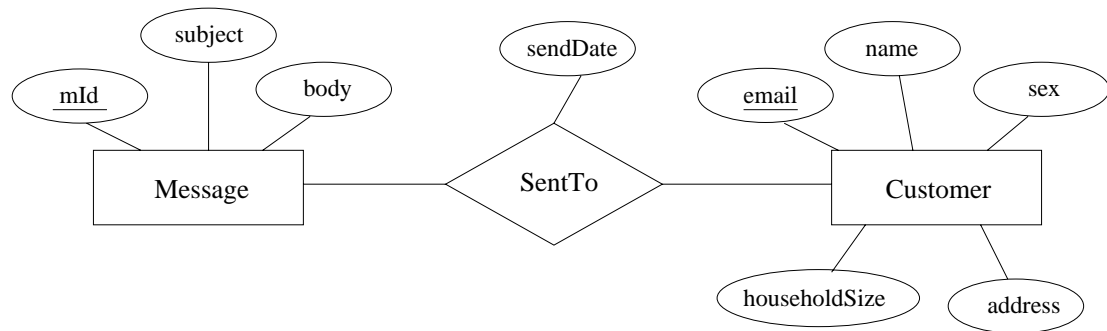


## Problems for Chapter 4: Conceptual Modeling of Databases with E-R and UML

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 domain

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

and change the domain of the attribute sex to: Sex

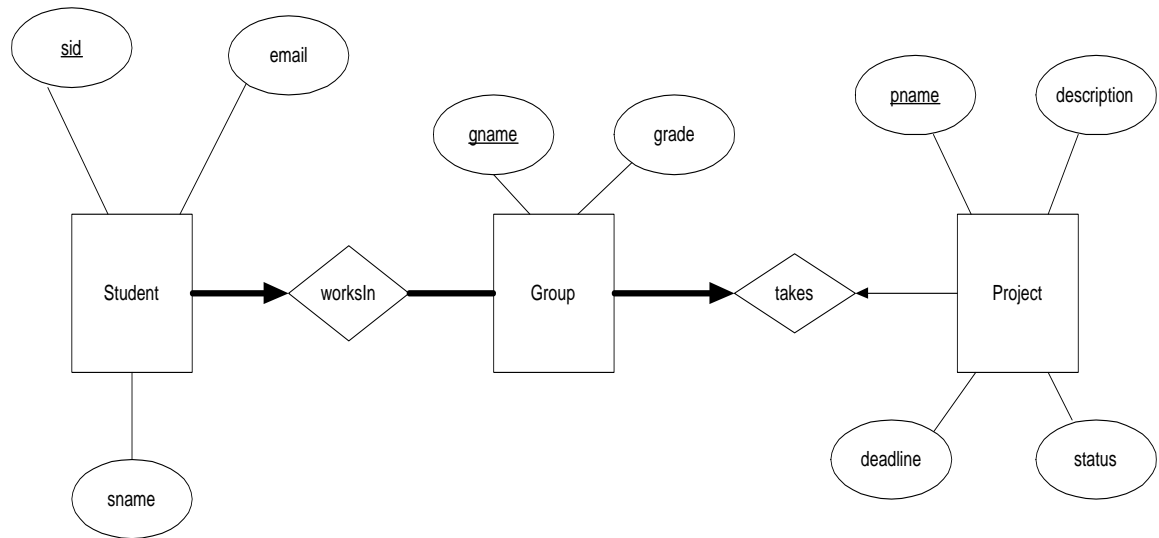
- (f) Change the primary key of the SEND table to mId. Since email will no longer be part of the primary key, it needs a NOT NULL constraint.  
(g) CREATE ASSERTION NoNeglectedCustomer  
CHECK (NOT EXISTS (  
 SELECT \* FROM CUSTOMER C  
 WHERE NOT EXISTS (  
 SELECT \* FROM SENTTo S  
 WHERE C.email = 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. That 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)  $\pi_{\text{name}} ( \sigma_{\text{dnumber}=666 \text{ AND } \text{gender}='male'}(\text{EMPLOYEE}) \bowtie_{\text{ssn}=\text{emplSSN}} ( \sigma_{\text{pname}='Catch22'}(\text{PROJECT}) \bowtie \sigma_{\text{hours}>13}(\text{WORKSON}) ) )$
- (b)  $\pi_{\text{name}} ( \text{EMPLOYEE} \bowtie_{\text{supervisorSSN}=\text{ssn}} (\text{EMPLOYEE} \bowtie_{\text{supervisorSSN}=\text{ssn2}} \sigma_{\text{name2}='JoePublic'} (\text{EMPLOYEE}[\text{ssn2}, \text{name2}, \text{gender2}, \text{address2}, \text{salary2}, \text{supervisorSSN2}, \text{dnumber2}]) ) )$
- (c)  $\pi_{\text{dmname}}(\text{DEPARTMENT}) - \pi_{\text{dmname}}(\text{DEPARTMENT} \bowtie \text{EMPLOYEE})$
- (d)  $\pi_{\text{name}}(\sigma_{\text{dnumber}=123}(\text{EMPLOYEE})) - \pi_{\text{name}} ( \sigma_{\text{dnumber}=123}(\text{EMPLOYEE}) \bowtie_{\text{ssn}=\text{emplSSN}} ( \sigma_{\text{plocation}='NYC'}(\text{PROJECT}) \bowtie \sigma(\text{WORKSON}) ) )$

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)  $R \cup S$ , (ii)  $R \cap S$  (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  $b_1, \dots, b_m$ , and R has in addition attributes  $a_1, \dots, a_n$ .

**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.
- 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.
  - Express the above query in SQL.
  - 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.)
  - 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:**

- $\pi_{name, name2, address}(\sigma_{householdSize=2}(USER) \bowtie_{email \neq email2 \text{ AND } address=address2} \sigma_{householdSize=2}(USER)[email2, name2, address2, householdSize2])$
- ```

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

```
- ```

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
ORDER BY householdSize

```
- ```

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 |                               |
|-------|-------|-------------------------------|
| 100   | 3     | //3 people got 100            |
| 90    | 20    | //20 people got between 90-99 |
| 80    | 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  $\langle 2, 0 \rangle$ . 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:**

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

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  $X \rightarrow Y$  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  $X \rightarrow Y$  such that
  - (1)  $Y$  contains all attributes of the schema, and
  - (2) There is no functional dependency  $X' \rightarrow Y$  for any subset  $X'$  of  $X$ .

2. Consider a schema  $S$  with functional dependencies:  
 $\{A \rightarrow BC, C \rightarrow FG, E \rightarrow HG, G \rightarrow A\}$ .
- (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)  $\{A \rightarrow F, E \rightarrow AF\}$  or  $\{A \rightarrow F, E \rightarrow A, E \rightarrow F\}$  or  $\{A \rightarrow F, E \rightarrow A\}$
- (c) No,  $A$  is not a superkey.  $R1 = \{AF, A \rightarrow F\}$  and  $R2 = \{EA, E \rightarrow A\}$ . 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              |
|--------------------|--------------------|
| <hr/>              | <hr/>              |
| $A \rightarrow B$  | $A \rightarrow C$  |
| $C \rightarrow A$  | $C \rightarrow B$  |
| $AB \rightarrow C$ | $CB \rightarrow A$ |

**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 BCGHMOVY:

$W \rightarrow V$   
 $WY \rightarrow BV$   
 $WC \rightarrow V$   
 $V \rightarrow B$   
 $BG \rightarrow M$   
 $BV \rightarrow Y$   
 $BYH \rightarrow V$   
 $M \rightarrow W$   
 $Y \rightarrow H$   
 $CY \rightarrow W$

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

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

$W \rightarrow V$   
 $V \rightarrow B$   
 $BG \rightarrow M$   
 $V \rightarrow Y$   
 $BY \rightarrow V$   
 $M \rightarrow W$   
 $Y \rightarrow H$   
 $CY \rightarrow W$

**Lossless 3NF:**  $(WV; \{W \rightarrow V\}), (VBY; \{BY \rightarrow V, V \rightarrow Y, V \rightarrow B\}),$   
 $(BGM; \{BG \rightarrow M\}), (MW; \{M \rightarrow W\}), (YH; \{Y \rightarrow H\}), (CYW; \{CY \rightarrow W\}).$

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

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

We decompose BGM with respect to  $M \rightarrow B$  into  $(BM; \{M \rightarrow B\})$  and  $(GM; \{\})$ , thereby losing the FD  $BG \rightarrow M$ .

Similarly CYW is decomposed using  $W \rightarrow Y$  into  $(WY; \{W \rightarrow Y\})$  and  $(WC; \{\})$ , losing  $CY \rightarrow W$ .

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

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

and the following multivalued dependencies:

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

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:

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

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

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

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

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

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

The decomposition is not dependency preserving. For instance, the FD  $A \rightarrow B$ , 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:

$$\begin{aligned} DE &\rightarrow F \\ BC &\rightarrow AD \\ FD &\rightarrow G \\ F &\rightarrow DE \\ D &\rightarrow E \end{aligned}$$

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

**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'.  $D \rightarrow F$
- 2'.  $BC \rightarrow A$
- 3'.  $BC \rightarrow D$
- 4'.  $F \rightarrow G$
- 5'.  $F \rightarrow D$
- 6'.  $F \rightarrow E$
- 7'.  $D \rightarrow E$

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

- 1'.  $D \rightarrow F$
- 2'.  $BC \rightarrow A$
- 3'.  $BC \rightarrow D$
- 4'.  $F \rightarrow G$
- 5'.  $F \rightarrow D$
- 7'.  $D \rightarrow E$

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  $BCAD^+ = BCDEFG$ , *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 be 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. These 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 DOWWithProgram
      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;
```

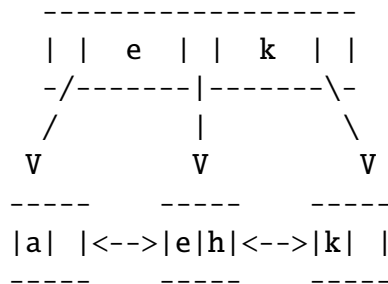
## Problems for Chapter 9: Physical Data Organization and Indexing

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:

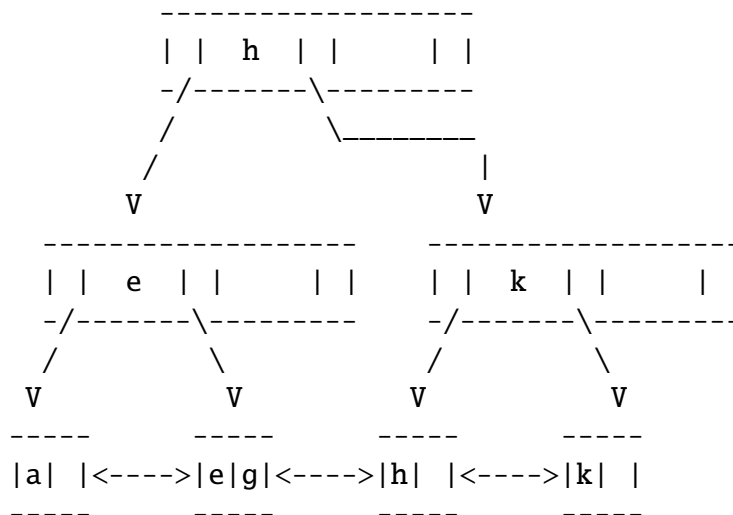
- (a) Minimize latency, and reduce the number of page transfers. Store pages containing related information close together on disk, and keep cache of recently accessed pages in main memory.
- (b) 500, 9 ( $=\lceil \log_2 500 \rceil$ ), 2 (the index has 500 pages, so the entire index fits in one page; 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:**

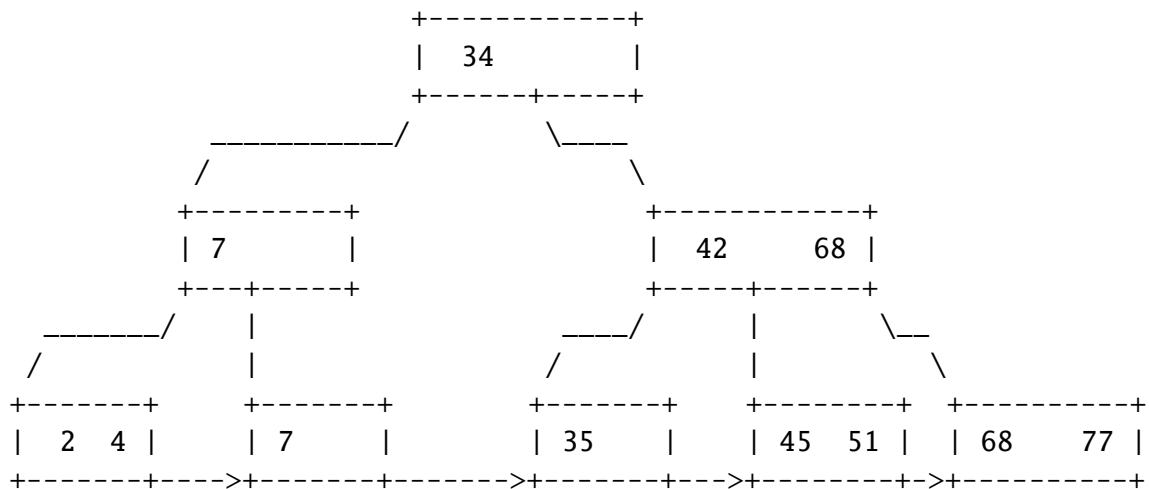
- (a) Supports efficient range search, and partial key search. Is faster if good hash function can be used to eliminate overflow chains.  
 (b) The root node will be split and the tree will add one level:



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.

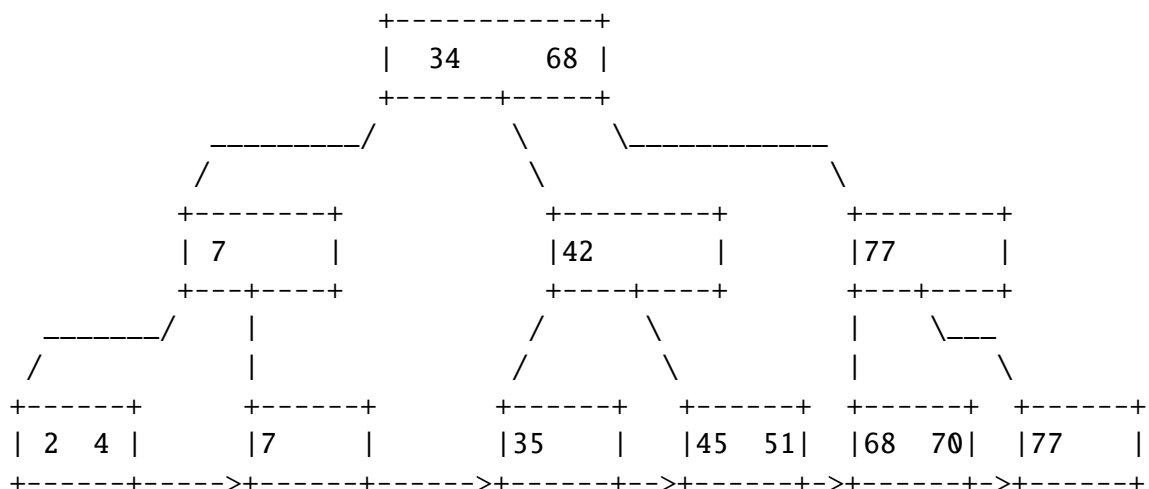
- Assuming that the index is *unclustered*, what is the maximal size (measured in data records) of a file that can be indexed by the depicted B+-tree?
- Same question, but assume that the index is now *clustered*.
- Show the B+-tree after inserting a new record with search key value 70.
- Show the tree 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.)

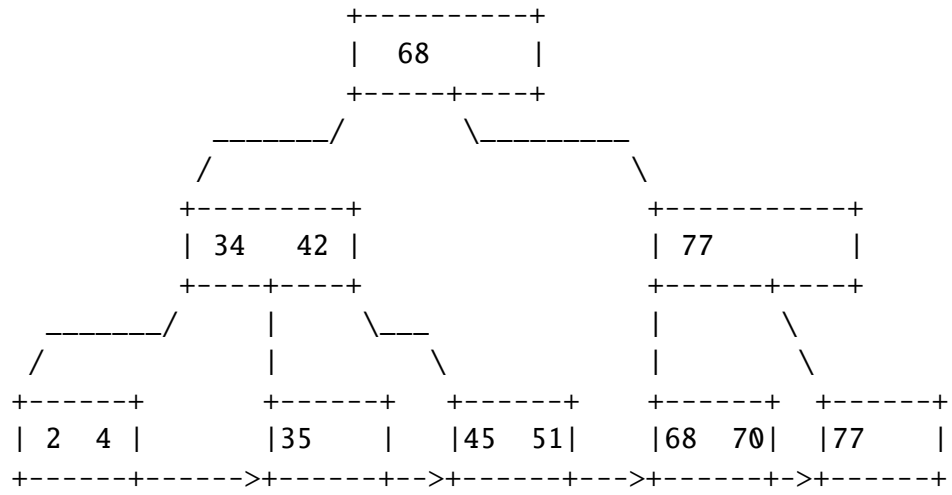


### Solution:

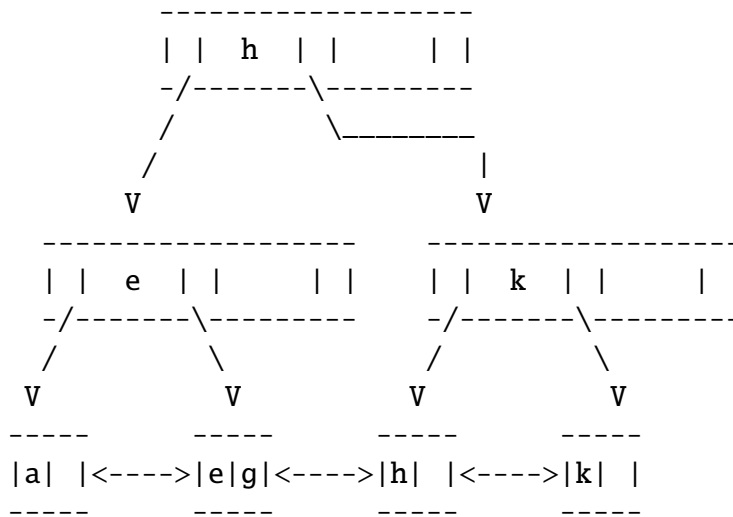
- Unclustered index: Each leaf entry can index only a single data record, so the max file size is 8 data records.
- Clustered index: the tree can index 8 *pages*, which can contain up to 80 data records.
- Insertion of 70:



(c) Deletion of 7:



4. Consider the following B+ tree:



- How many pages of actual data file does the tree provide index for?
- 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?
- At least how many keys must be inserted so that the left node in the middle level would split?
- At most how many keys can be inserted without splitting the root node?

**Solution:**

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

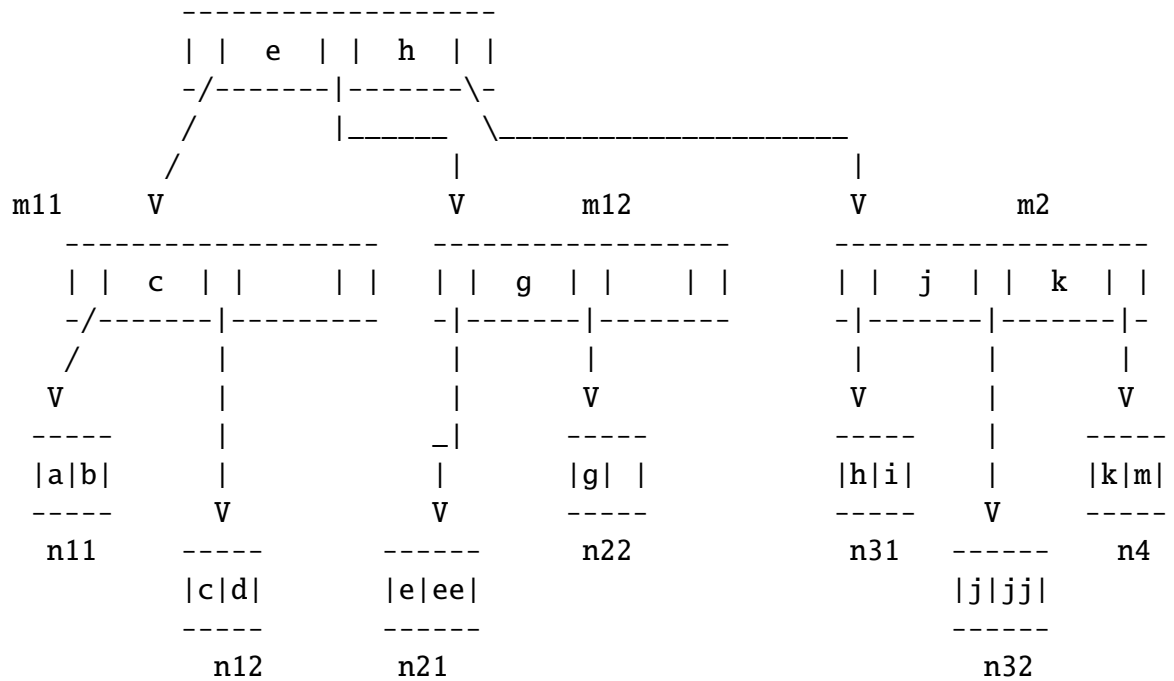
(d) 13 keys. This is a tedious, but simple count. As in (a), we should be adding keys as conservatively as possible trying to delay splits as long as possible.

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.

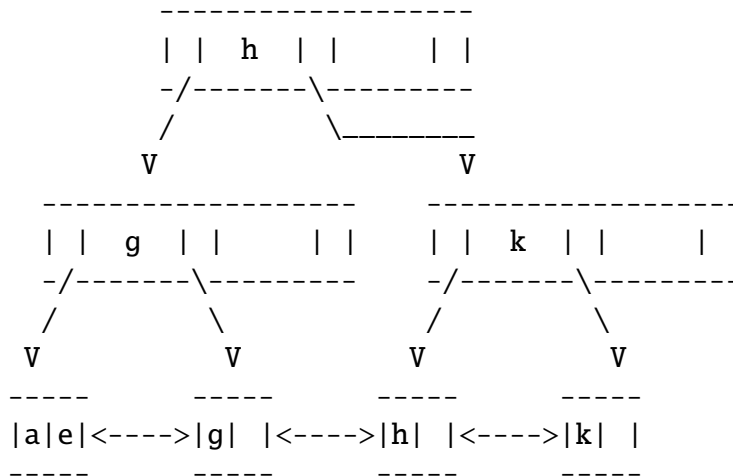
Add 1 key to n12. This will split it up into n121, n122 and fill up m11. n121 will be full, but n122 will have one free slot.

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.



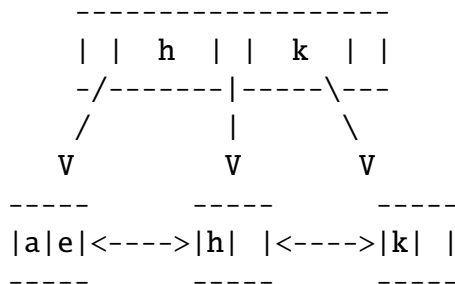
5. Consider the B<sup>+</sup> tree below.



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  $h_k(v) = h(v) \bmod 2^k$  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  $\sigma_{A=18}(s)$ .
- (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  $\text{ceiling}(1.2 + 35) = 37$ .
- (b) If the index is clustered, then 37 tuples can at most span 2 pages, so the I/O cost would be  $\text{ceiling}(1.2+2)=4$ .

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  $\langle \text{StudId}, \text{Grade} \rangle$  that sends two identical tuples in  $\pi_{\text{StudId}, \text{Semester}}(\text{TRANSCRIPT})$  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:  $\langle 111111111, \text{EE101}, \text{F1997}, \text{A} \rangle$  and  $\langle 111111111, \text{MAT123}, \text{F1997}, \text{B} \rangle$ . Since these tuples have identical projections on the attributes StudId, Semester, they become one in the projected relation

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

Consider now the following hash function on StudId, Grade:

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

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

5. Estimate the cost of  $\mathbf{r} \bowtie \mathbf{s}$  using

- (a) Sort-merge join
- (b) Block nested loops

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

**Solution:**

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

- (a) Sorting  $\mathbf{r}$  will take  $2 \cdot 50 \cdot \log_{21} 50 \approx 200$ . Sorting  $\mathbf{s}$  will take  $2 \cdot 500 \cdot \log_{21} 500 \approx 3000$ . 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  $\mathbf{r}$  is scanned in the outer loop, the cost is  $50 + (50/20) \cdot 500 = 1300$ .

6. Consider the expression

$$\sigma_{\text{StudId}=666666666 \wedge \text{Semester}='F1995' \wedge \text{Grade}='A'}(\text{TRANSCRIPT})$$

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 would 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  $\mathbf{s}$ . (Do the computation for both clustered and unclustered index.)

where  $\mathbf{r}$  occupies 2,000 pages, 20 tuples per page,  $\mathbf{s}$  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  $\mathbf{s}$  match each tuple in  $\mathbf{r}$ .

**Solution:**

- (a) Nested loops: scan  $\mathbf{r}$  and for each of its 40,000 tuples scan  $\mathbf{s}$  once. The result is

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

- (b) Block-nested loops: Scan  $\mathbf{s}$  once per each 400-page block of  $\mathbf{r}$ , 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  $\mathbf{s}$  is clustered or not. For the clustered case, all tuples of  $\mathbf{s}$  that match a tuple of  $\mathbf{r}$  are in the same disk block and require 1 page transfer (since we assumed that at most 5 tuples of  $\mathbf{s}$  match, they all fit in one disk block). We also need to search the index once per each tuple of  $\mathbf{r}$ . 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  $\mathbf{s}$  can be in different blocks. As before, assume that  $\mathbf{s}$  has at most 5 matching tuples per tuple in  $\mathbf{r}$ . 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)  $\pi_{\text{street}} \sigma_{\text{id}=\text{toAddrId} \text{ AND } \text{townState}=\text{'StonyBrookNY'} \text{ AND } \text{date}=\text{'05/14/02'}}(\text{Address} \times \text{TRIP})$
- (c)  $\pi_{\text{street}}(\sigma_{\text{townState}=\text{'StonyBrookNY'}}(\text{ADDRESS}) \bowtie_{\text{id}=\text{toAddrId}} \sigma_{\text{date}=\text{'05/14/02'}}(\text{TRIP}))$
- (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  $\mathbf{r}$  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<sup>+</sup> tree index on attribute B
- Attribute B has 1,000 distinct values in  $\mathbf{r}$
- Attribute C has 500 distinct tuples and an unclustered 3-level B<sup>+</sup> 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  $\mathbf{r}$ , 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<sup>+</sup> 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<sup>+</sup> 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  $\mathbf{R}$ ,  $\mathbf{S}$ , and  $\mathbf{T}$  *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:**

- $\mathbf{R}$  must have:  $B$  (because of the join)
- $\mathbf{S}$  must have:  $ACD$  (because of  $\pi_{ACD}$ )
- $\mathbf{T}$  must have:  $E$  (because of  $\pi_E$ )
- 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,  $\mathbf{T}$  should not have  $A$  and  $C$ , because  $\pi_{ACD}(\mathbf{S})$  clearly suggests that it is expected that  $\mathbf{S}$  will have the attribute  $A$  that will survive for the outermost projection  $\pi_A$  to make sense, and the attribute  $C$ , which should survive in order for the join with  $\mathbf{R}$  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  $\pi_A$  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  $\mathbf{S}$  and not of  $\mathbf{T}$ .
- (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 schema, where the keys are underlined:

ITEM(Name, Category)  
 STORE(Name, City, StreetAddr)  
 TRANSACTION(ItemName, StoreName, Date)

Here a tuple  $(i, s, d) \in \text{TRANSACTION}$  denotes the fact that item  $i$  bought at store  $s$  on date  $d$ .)

Consider the following query:

$$\pi_{\text{Category}, \text{City}}(\sigma_{\text{Date}='2004-12-15' \text{ AND } \text{City}='NYC'}(\text{ITEM} \bowtie_{\text{Name}=\text{ItemName}} \text{TRANSACTION} \bowtie_{\text{StoreName}=\text{Name}} \text{STORE}))$$

Show the three “most promising” relational algebra expressions that the query optimizer is likely to consider; then find the most efficient query plan and estimate its cost.

Assume 50 buffer pages and the following statistics and indices:

- ITEM: 50,000 tuples, 10 tuples/page.  
Index: Unclustered hash on Name.
- STORE: 1,000 tuples, 5 tuples/page; 100 cities.  
Index1: Unclustered hash index on Name.  
Index2: Clustered 2-level B<sup>+</sup> tree on City.
- TRANSACTION: 500,000 tuples, 25 tuples/page; 10 items bought per store per day. The relation stores transactions committed over a 50 day period.  
Index: 2-level clustered B<sup>+</sup> tree on the pair of attributes StoreName, Date.

**Solution:** The optimizer will consider the fully pushed and the two partially pushed expressions:

$$\pi_{\text{Category,City}}(\text{ITEM} \bowtie_{\text{Name=ItemName}} \sigma_{\text{Date='2004-12-15'}}(\text{TRANSACTION}) \bowtie_{\text{StoreName=Name}} \sigma_{\text{City='NYC'}}(\text{STORE}))$$

$$\pi_{\text{Category,City}}(\text{ITEM} \bowtie_{\text{Name=ItemName}} \sigma_{\text{Date='2004-12-15'}}(\text{TRANSACTION} \bowtie_{\text{StoreName=Name}} \sigma_{\text{City='NYC'}}(\text{STORE})))$$

$$\pi_{\text{Category,City}}(\text{ITEM} \bowtie_{\text{Name=ItemName}} \sigma_{\text{City='NYC'}}(\sigma_{\text{Date='2004-12-15'}}(\text{TRANSACTION}) \bowtie_{\text{StoreName=Name}} \text{STORE}))$$

The most efficient query plan would be to use the last partially pushed expression with the join order reversed:

$$\pi_{\text{Category,City}}(\text{ITEM} \bowtie_{\text{Name=ItemName}} \sigma_{\text{Date='2004-12-15'}}(\sigma_{\text{City='NYC'}}(\text{STORE}) \bowtie_{\text{StoreName=Name}} \text{TRANSACTION}))$$

In addition, we can combine the selection of Date with the inner join, as explained below.

The costs are computed as follows:

(a)  $\sigma_{\text{City='NYC'}}(\text{STORE})$  – selects 10 tuples (2 pages) using the 2-level B<sup>+</sup> tree index. Cost: 2+2=4.

(b) Join the result with TRANSACTION on StoreName using the 2-level B<sup>+</sup> tree index on TRANSACTION. Moreover, here we can combine the index nested loops join with selection on Date as follows. Note that the index is on the pair of attributes StoreName Date, while the join is only on the first attribute. However, we can combine each index search that is performed to compute the join with  $\sigma_{\text{Date}}$ . In this combination, every index search will be on the full index StoreName Date rather than just a prefix StoreName.

Since  $\sigma_{\text{City='NYC'}}(\text{STORE})$  has 10 tuples, 10 full-index selections on TRANSACTION will be performed during the combined join/selection computation. According to the statistics, 10 items are sold per store per day. Therefore, each selection will bring 10 tuples, 1 page. It takes 3 I/Os to bring those tuples. Since we need to perform 10 selections, the cost is 3\*10=30 and the result has 100 tuples.

(c) Join the result with ITEM using the unclustered hash index on Name. Since Name is a key, each hash brings 1 tuple and costs 1.2 (the cost of the hash) + 1 (the cost of retrieving the matching tuple) = 2.2 I/Os. If we do it 100 times for each tuple in the result of (2), we will spend 220 I/Os.

Therefore, the total is 4+30+220=254 I/Os.

5. 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<sup>+</sup> 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<sup>+</sup> 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.

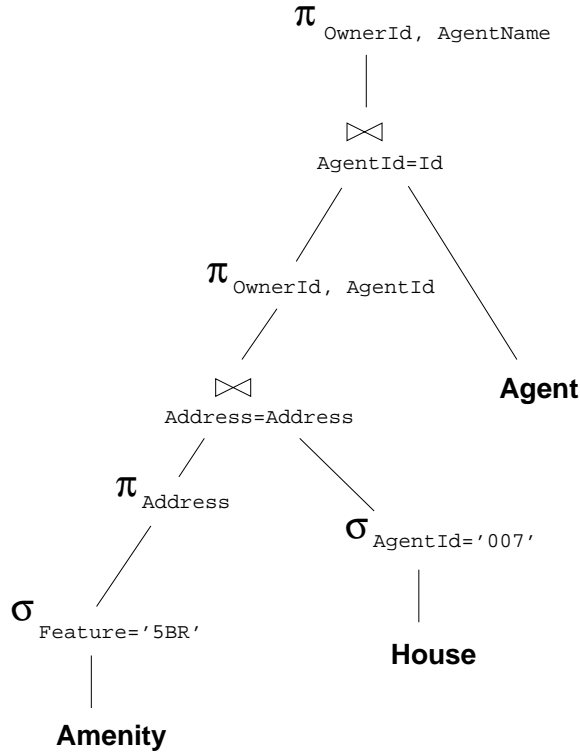


Figure 1:

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 lose 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.,  $2 \times 5$ ; 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 ( $5 \times 10$ ), 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)  $\sigma_{\text{Course}='CS305'}(\text{Took}) \bowtie_{\text{StudId}=\text{Id}} \sigma_{\text{Major}='EE'}(\text{Student})$

- (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.

2. The table Employee(Id, Name, DeptId, Salary) has Id as its primary key. What index would you choose to enhance the performance of the following statement?

```
SELECT E.Name  
FROM Employee E  
WHERE E.Salary > 100000 AND E.DeptName = 'accounting'
```

**Solution:**

The index on the primary key is of no help. An additional index on either Salary or DeptName is needed depending on which is more selective. If the number of employees in accounting is smaller than the number earning over \$100000 then a hash or B<sup>+</sup> tree index on DeptName would be appropriate. Performance can be further improved by making the index clustered, so that all employees in the same department are in the same hash bucket or are consecutive if a B<sup>+</sup> structure is used. In that case the index on Id must be unclustered, but that should not imply a performance penalty since queries involving Id will generally not involve a range search and hence will return a single row. If accounting is a large department then a B<sup>+</sup> tree index on Salary would be appropriate since a range search is needed. The same considerations with respect to clustering apply.

3. The table Employee(Id, Name, DeptId, Salary, ...) has Id as its primary key. Describe a situation that might justify the use of an index covering strategy.

**Solution:**

Assume that the table has a large number of attributes, so that each row occupies a significant amount of space. Assume also that a clustered index exists on certain attributes to enhance the performance of a particular query. For example, a clustered index on Salary might be required if the query

```
SELECT E.Name
FROM Employee E
WHERE E.Salary > :sal
```

were frequently executed. Suppose another frequently asked query is

```
SELECT E.Name
FROM Employee E
WHERE E.DeptId = :dept
```

Since the number of employees in a department might be large it is desirable to support the query with a clustered index on DeptId, but this is not possible since one clustered index already exists. An unclustered B<sup>+</sup> tree index on (DeptId, Name) contains all the information needed to respond to the query without consulting the table itself. Leaf level entries are sorted on DeptId and hence can be efficiently retrieved. Note that since the number of characters needed to store the values of these attributes is small, the index itself is small and only a few pages will need to be retrieved to satisfy the query.

#### 4. The SELECT statement

```
SELECT A
FROM T
WHERE P
```

accesses a table, T, with attributes A, B, C and K, where K is the primary key, and P is a predicate. The values of all attributes are randomly chosen over the integer domain. Since the table is updated frequently, there is a limitation of at most two indexes. In each of the following parts choose the indexes (type, search key, clustered or unclustered) to optimize the SELECT statement and maintain the primary key constraint when the value of P is as specified. In each case give the query plan you expect the DBMS to choose for the SELECT and the reason you think that plan is best. Do not use index covering.

- (a) P is (B = 10)

**Solution:**

clustered index on B; unclustered on K

search on B=10, scan (B<sup>+</sup> tree) or search bucket (hash) for all satisfying rows

- (b) P is (B > 10)

**Solution:**

clustered B<sup>+</sup> tree on B, unclustered on K

rows are ordered on B, search for B=10 and scan from that point

- (c) P is (B > 10 AND C = 5)

**Solution:**

Fewer rows satisfy C=5 than B>10 (values are randomly chosen). Hence use clustered index on C, clustered on K; hash or B<sup>+</sup> tree OK for both.

Fetch rows satisfying C=5 and return those having B>10

- (d) P is (B > 10) and the WHERE clause is followed by the clause ORDER BY A

**Solution:**

Clustered B<sup>+</sup> tree on A, unclustered hash or B<sup>+</sup> on K. Scan entire file, discard rows return rows with B>10. This is better than clustering on B since roughly half of the rows satisfy B>10 and they would require sorting.

- (e) P is (B > 10 AND C = 5)

**Solution:**

Fewer rows satisfy C=5 than B>10 (values are randomly chosen). Hence use clustered index on C, clustered on K; hash or B<sup>+</sup> tree OK for both.

Fetch rows satisfying C=5 and return those having B>10

- (f) P is (B > 10) and the WHERE clause is followed by the clause ORDER BY A

**Solution:**

Clustered B<sup>+</sup> tree on A, unclustered hash or B<sup>+</sup> on K. Scan entire file, discard rows return rows with B>10. This is better than clustering on B since roughly half of the rows satisfy B>10 and they would require sorting.

5. Consider a database with two tables: EMP, with attributes empName and divName and Div with attributes divName and building. You are required to write a select statement that lists the names of all employees in divisions that have space in building A. (A division might have space in several buildings.)

(a) Write the statement using a join on the two relations.

**Solution:**

```
SELECT E.empName
FROM EMP E, Div D
WHERE E.divName = D.divName AND D.building = 'A'
```

(b) Write the statement in a form that uses a subquery, but no join.

**Solution:**

```
SELECT E.empName
FROM EMP E
WHERE E.divName IN (
    SELECT D.divName
    FROM Div D
    WHERE D.building = 'A')
```

(c) Assuming no useful indexes exist, comment on the efficiency of executing the two forms of the query. For example, under what conditions (e.g., on the number of rows in each relation, the number of divisions with offices in building A, the number of employees in a division) would you expect one form of the query to outperform the other.

**Solution:**

Assume  $n$  pages in EMP and  $m$  pages in Div. A query plan for the first statement might do a scan of Div for each page of EMP, requiring  $n \times m$  page transfers.

Since the subquery is not correlated, a query plan for the second might first evaluate the subquery to identify divisions having space in building A. The scan requires  $m$  page transfers and results in an intermediate relation of  $r$  pages where  $\max(r) = m$ . It then does  $n \times r$  page transfers to combine rows in EMP with the intermediate relation. Hence the total number of operations is  $m + n \times r$  and if  $r$  is much less than  $m$  - which is likely - the second statement can be evaluated more efficiently.

(d) In a more general situation, the identity of the building will be input as a parameter. What measure might you take (denormalization, use of an index) to speed the execution of first statement?

**Solution:**

Denormalizing by adding a building attribute to EMP doesn't work, since an employee in building B might be working for a division that also has space in building A. Indexing Div on divName allows the use of an index-nested loop join. The scan of Div can be avoided to form the intermediate relation.

## Problems for Chapter 13: Relational Calculus, Deductive Databases, and Visual Query Languages

1. Write the following query using TRC and DRC: Find the names of all brokers who have made money in all accounts assigned to them. Assume the following schema:

BROKER(Id, Name)  
ACCOUNT(Acct#, BrokerId, Gain)

**Solution:**

TRC:     {B.Name | BROKER(B) AND  
                   $\forall A \in \text{ACCOUNT} (A.\text{BrokerId} = B.\text{Id} \rightarrow A.\text{Gain} > 0)$ }

DRC:     {Name |  $\exists \text{BrokerId} (\text{BROKER}(\text{BrokerId}, \text{Name}) \text{ AND}$   
                   $\forall \text{Acc\#} \forall \text{Gain} (\text{ACCOUNT}(\text{Acc\#}, \text{BrokerId}, \text{Gain}) \rightarrow \text{Gain} > 0))$ }

2. Express the following query in TRC and DRC: Find all courses in department MGT that were taken by all students. Assume the following schema:

COURSE(CrsCode, DeptId, CrsName, Descr)  
TRANSCRIPT(StudId, CrsCode, Semester, Grade)  
STUDENT(Id, Name, Status, Address)

**Solution:**

TRC:     {C.CrsCode, C.CrsName | COURSE(C) AND C.DeptId = 'MGT' AND  
                                   $\forall S \in \text{STUDENT } \exists R \in \text{TRANSCRIPT}$   
                                  (R.StudId = S.Id AND R.CrsCode = C.CrsCode) }

DRC:     {CrscCode, CrsName |  $\exists \text{Descr (COURSE('MGT', CrsCode, CrsName, Descr))}$   
                                  AND  $\forall \text{Id} \in \text{STUDENT.Id } \exists \text{Semester} \exists \text{Grade}$   
                                  (TRANSCRIPT(Id, CrsCode, Semester, Grade) ) }



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

ACTOR(Id, Name)

MOVIE(Id, Title, DirectorName)

PLAYED(ActorId, MovieId)

Write the following query in **tuple** relational calculus: “Find all actors (*Id*, *Name*) who played in every movie produced by the director Joe Public.”

**Solution:**

$$\{A.Id, A.Name \mid \text{ACTOR}(A) \text{ AND } \forall M \in \text{MOVIE}(\text{M.DirectorName} = \text{'Joe Public'} \rightarrow \exists P \in \text{PLAYED}(\text{M.Id} = \text{P.MovieId AND } A.Id = \text{P.ActorId}) ) \}$$

4. Consider a relation `DIRECTFLIGHT(StartCity, DestinationCity)` that lists all direct flights among cities. Use the recursion facility of SQL:1999 to write a query that finds all pairs  $\langle city_1, city_2 \rangle$  such that there is an *indirect* flight from  $city_1$  to  $city_2$  with at least two stops in-between.

**Solution:**

The simplest way to do this is to compute `INDIRECTFLIGHT` similarly to `INDIRECTPREREQVIEW`:

```
CREATE RECURSIVE VIEW INDIRECTFLIGHT(From, To) AS
  SELECT * FROM DIRECTFLIGHT
  UNION
  SELECT D.StartCity, I.To
  FROM DIRECTFLIGHT D, INDIRECTFLIGHT I
  WHERE D.DestinationCity = I.From
```

Then we can compute all flights with just one stop — `FLIGHT1` — and then subtract `DIRECTFLIGHT` and `FLIGHT1` from `INDIRECTFLIGHT`.

One might be tempted to first create a recursive view `INDIRECTFLIGHT2(From, To, NumberOfStops)` and then select the flights with `NumberOfStops > 1`.

```
CREATE RECURSIVE VIEW INDIRECTFLIGHT2(From, To, Stops) AS
  SELECT D.StartCity, D.DestinationCity, 0
  FROM DIRECTFLIGHT D
  UNION
  SELECT D.StartCity, I.To, I.Stops+1
  FROM DIRECTFLIGHT D, INDIRECTFLIGHT2 I
  WHERE D.DestinationCity = I.From
```

However, this recursive definition has a problem: because we keep incrementing the number of stops with each iteration, the evaluation process will not terminate. (Check that the termination condition will never be true!)

## Problems for Chapter 14: 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
```

2. Consider the following schema:

ACTOR(Id, Name)  
MOVIE(Id, Title, DirectorName)  
PLAYED(ActorId, MovieId)

- (a) Represent this schema using the object-relational extensions of SQL:1999/2003. Use set-valued attributes and object-oriented design.

**Solution:**

```
CREATE TYPE PERSONTYPE AS (  
    Name CHAR(30)  
)  
CREATE TYPE ACTORTYPE UNDER PERSONTYPE AS (  
    PlaysIn REF(MOVIE TYPE) MULTISSET SCOPE MOVIES  
)  
CREATE TYPE MOVIE TYPE AS (  
    Title CHAR(100),  
    Director REF(PERSON),  
    Stars REF(ACTORTYPE) MULTISSET SCOPE ACTORS  
)  
CREATE TABLE Actors OF ACTORTYPE  
CREATE TABLE Movies OF MOVIE TYPE
```

- (b) Use the schema constructed in subproblem (a) to formulate the following query using SQL:2003: *For every actor, ac, count the number of movies directed by Joe Schmuck where the actor ac played a role.*

**Solution:**

```
SELECT M.Stars->Name, count(N.Title)  
FROM MOVIES M, UNNEST(M.Stars) S, Movies N  
WHERE S IN N.Stars AND N.Director->Name = 'Joe Schmuck'  
GROUP BY M.Stars->Name
```

3. Consider an ACCOUNT class and a TRANSACTIONACTIVITY class in a banking system.

- (a) Posit ODMG ODL class definitions for them. The ACCOUNT class must include a relationship to the set of objects in the TRANSACTIONACTIVITY class corresponding to the deposit and withdraw transactions executed against that account.

**Solution:**

```
class ACCOUNT {
    attribute Integer AcctId;
    relationship Set< PERSON> Owner;
    relationship Set< TRANSACTIONACTIVITY> Transactions
        inverse TRANSACTIONACTIVITY::ActivityAccount;
}

class TRANSACTIONACTIVITY {
    // assume type is some integer code
    attribute enum TransType {deposit,withdraw} Type;
    attribute Float Amount;
    attribute Date ActivityDate;
    relationship ACCOUNT ActivityAccount
        inverse ACCOUNT::Transactions;
}
```

- (b) Give an example of an object instance satisfying that description.

**Solution:**

```
(#123, [12345,
    {(#p4345, [123456789, "John Doe"]),
    (#p0987, [654345643, "Mary Doe"])},
    {(#t435, [withdraw, 58.34, 2001-3-4, #123]),
    (#t8132, [deposit, 231.99, 2001-4-5, #123])}]
])

....
```

- (c) Give an example of an OQL query against that database, which will return the account numbers of all accounts for which there was at least one withdrawal of more than \$10,000.

**Solution:**

```
SELECT A.AcctId
FROM ACCOUNTEXT A, A.Transactions T
WHERE T.Type = withdraw AND T.Amount > 10000
```

4. Write an OQL query that, for each major, computes the number of students who have that major. Use the following partial definition of STUDENT:

```
class STUDENT extends PERSON
  (extent STUDENTEXT)
  ...
  attribute Set<String> Major;
  relationship Set<Course> Enrolled;
    inverse COURSE::Enrollment;
  ...

class COURSE : Object
  ...
  relationship Set<Student> Enrollment;
    inverse STUDENT::Enrolled;
  ...
```

**Solution:**

```
SELECT Major: M, count: count(S2.Id)
FROM STUDENTEXT S, STUDENTEXT S2, S.Major M
WHERE M IN S2.Major
GROUP BY M
```

We could also write this query using nested queries as follows:

```
SELECT DISTINCT Major: M,
    count: count( SELECT S2.Id
                  FROM STUDENTEXT S2
                  WHERE S2.Major = S.Major AND M IN S2.Major )
FROM STUDENTEXT S
```

## Problems for Chapter 15: 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"
    CsrName="Transaction Processing" CreditHours="3">
    <Prerequisite CsrCode="CS305" EnforcedSince="2000/08/01"/>
    <Prerequisite CsrCode="CS219" EnforcedSince="2001/01/01"/>
  </Course>
  <Course>
    .....
  </Course>
  .....
</Courses>
```

An appropriate DTD would be:

```
<!DOCTYPE Courses [
  <!ELEMENT Courses (Course*)>
  <!ELEMENT Course (Prerequisite*)>
  <!ATTLIST Course
    CsrCode ID #REQUIRED
    DeptId CDATA #IMPLIED
    CsrName CDATA #REQUIRED
    CreditHours CDATA #REQUIRED >
  <!ATTLIST Prerequisite
    CsrCode 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. Consider XML documents of the form below, where actor's name and movie's title uniquely identify actors and movies, respectively.  
Write the corresponding XML schema including the key and foreign key constraints.

```
<MovieLand>
  <Actor>
    <Name>...</Name>
    <PlayedIn>
      <MovieTitle>...</MovieTitle>
      ...
      <MovieTitle>...</MovieTitle>
    </PlayedIn>
  </Actor>
  ...
  <Actor>
    ...
  </Actor>

  <Movie>
    <Title>...</Title>
    <Actors>
      <ActorName>...</ActorName>
      ...
      <ActorName>...</ActorName>
    </Actors>
  </Movie>
  ...
  <Movie>
    ...
  </Movie>
</MovieLand>
```

**Solution:** We will use only anonymous types in this solution.

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:this="http://myschema.name"
        targetNamespace="http://myschema.name">

  <element name="MovieLand">
    <complexType>
      <sequence>

        <element name="Actor" minOccurs="0" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="Name" type="String"/>
              <element name="PlayedIn">
                <complexType>
                  <element name="MovieTitle" type="String"
                        minOccurs="0" maxOccurs="unbounded"/>
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>

        <element name="Movie" minOccurs="0" maxOccurs="unbounded">
          <complexType>
            <sequence>
              <element name="Title" type="String"/>
              <element name="Actors">
                <complexType>
                  <element name="ActorName" type="String"
                        minOccurs="0" maxOccurs="unbounded"/>
                </complexType>
              </element>
            </sequence>
          </complexType>
        </element>

      </sequence>
      <!-- include key specifications here -- next page -->
    </complexType>
  </element>
</schema>
```

Specification of keys and foreign keys.

```
<key name="ActorKey">
  <selector xpath="Actor" />
  <field xpath="Name" />
</key>
<keyref name="ActorFK" refer="this:MovieKey">
  <selector xpath="Actor/PlayedIn"/>
  <field xpath="MovieTitle"/>
</keyref>

<key name="MovieKey">
  <selector xpath="Movie" />
  <field xpath="Title" />
</key>
<keyref name="MovieFK" refer="this:ActorKey">
  <selector xpath="Movie/Actors"/>
  <field xpath="ActorName"/>
</keyref>
```

5. 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
```

6. Use XSLT to transform the document in Figure 17.15 into a well-formed XML document that contains the list of `Student` elements such that each student in the list took a course in spring 1996.

**Solution:**

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xsl:version="1.0">
  <xsl:template match="/">
    <StudentListS1996>
      <xsl:apply-templates/>
    </StudentListS1996>
  </xsl:template>
  <xsl:template match="//Student">
    <xsl:if test="../CrsTaken/@Semester='S1996'">
      <xsl:copy-of select="."/>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

7. Write an XSLT stylesheet that traverses the document tree and, ignoring attributes, copies the elements and *doubles* the text nodes. For instance, `<foo a="1">the<best/>bar</foo>` would be converted into `<foo>thethe<best/> barbar</foo>`.

**Solution:**

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xsl:version="1.0">
  <xsl:template match="/*">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>
  <xsl:template match="text()">
    <xsl:value-of select=".">
    <xsl:value-of select=".">
  </xsl:template>
</xsl:stylesheet>
```



8. Write an XSLT stylesheet, which takes a document and transforms it as follows. The element nodes and attribute nodes are copied as is. However, the text nodes that are children of an element node are deleted and replaced with an attribute called `text`. The value of the `text` attribute is a string that is a concatenation of the strings represented by the deleted text nodes. For instance,

```
<aaa bbb="1">  
  abc <dddd eee="2">cde</dddd> fgh  
  <fff gg="3" />  
</aaa>
```

should become

```
<aaa bbb="1" text="abcfgh">  
  <dddd eee="2" text="cde"></dddd>  
  <fff gg="1" text="" />  
</aaa>
```

*Hint:* An attribute `text` can be created using the XSLT instruction

```
<xsl:attribute name = "text">  
  <!-- Content -->  
</xsl:attribute>
```

similarly to the way new elements are created in XSLT using the `xsl:element` instruction. A concatenation of all t-children of an element can be obtained with the XPath selector `text()`.

**Solution:** The XSLT transformation is as follows:

```
<xsl:template match="text()">
  <!-- ignore -->
</xsl:template>

<xsl:template match="*">
  <xsl:copy>
    <xsl:attribute name="text()">
      <xsl:value-of select="concat(./text())"/>
    </xsl:attribute>

    <xsl:apply-templates/>
    <xsl:apply-templates select="@*" />

  </xsl:copy>
</xsl:template>

<xsl:template match="@*">
  <xsl:copy-of select="."/>
</xsl:template>
```

9. Use the document structure described in exercise 17.21 to formulate the following queries in XQuery:

- (a) List all classes (identified by a course code and semester) where every student received a B or higher.

**Solution:**

```
DECLARE FUNCTION classes() AS element()* {
  FOR $c IN document("http://xyz.edu/transcript.xml")//tuple
  RETURN
    <class> $s//CrsCode, $c//Semester </class>
}

<EasyClasses>
(
  FOR $c IN distinct(classes())
  LET $grades :=
    document("http://xyz.edu/transcript.xml")
      //tuple[CrsCode/@value=$c/CrsCode/@value
        and Semester/@value=$c/Semester/@value]
      /Grade/@value
  WHERE EVERY $g IN $grades
    SATISFIES numericGrade($g) >= numericGrade("B")
  RETURN $c
)
</EasyClasses>
```

- (b) List all students who never received less than a B.

**Solution:**

```
<GoodStudents>
(
  FOR $sid IN
    distinct(document("http://xyz.edu/transcript.xml")//tuple/StudId/@value)
    $s IN document("http://xyz.edu/student.xml")
      //tuple[Id/@value=$sid]
  LET $grades :=
    document("http://xyz.edu/transcript.xml")
      //tuple[StudId/@value=$sid]/Grade/@value
  WHERE EVERY $g IN $grades
    SATISFIES numericGrade($g) >= numericGrade("B")
  RETURN $s
)
</GoodStudents>
```

10. Write an XQuery function that traverses a document and computes the maximum branching factor of the document tree, i.e., the maximal number of children (text or element nodes) of any element in the document.

**Solution:**

```

NAMESPACE xsd = "http://www.w3.org/2001/XMLSchema"
DECLARE FUNCTION maxBranching($n) RETURNS xsd:integer
{
    LET $children := $n/node()
    LET $maxChildBranching :=
        max{ FOR $m IN $children RETURN maxBranching($m) }
    LET $childCount := count($children)
    RETURN
        IF $childCount >= $maxChildBranching
        THEN $childCount
        ELSE $maxChildBranching
}
```

Recall that the XPath function `node()` matches every *e*- or *t*-node.

11. Consider a document of the form

```
<stores>
  <store name="..." city="...">
    <item name="..." price="...">
    <item name="..." price="...">
    <item name="..." price="...">
    ... ..
  </store>
  <store name="..." city="...">
    <item name="..." price="...">
    <item name="..." price="...">
    <item name="..." price="...">
    ... ..
  </store>
  ... ..
</stores>
```

Assume that name and city uniquely determine a store and within each store there is at most one item with a given name. Write the following query using XQuery:

*Find the stores with the **lowest** average price of items among all the stores in New York City.*

Return the names of all such stores.

**Solution:**

```
<Result>
  for $s in document("...")/store[@city="NYC"]
  where fn:avg($s//@price) =
    min( for $s1 in document("...")/store[@city="NYC"]
          return fn:avg($s1//@price) )
  return <storeName> { $s/name } </storeName>
</Result>
```

## Problems for Chapter 16: Distributed Databases

1. Show that the semijoin operation is not commutative, that is,  $\mathbf{T}_1 \bowtie_{\text{join-condition}} \mathbf{T}_2$  is not the same as  $\mathbf{T}_2 \bowtie_{\text{join-condition}} \mathbf{T}_1$ .

**Solution:**

Informally, the semijoin of  $\mathbf{T}_1$  and  $\mathbf{T}_2$ , based on a join condition, consists of the tuples of  $\mathbf{T}_1$  that participate in the join with  $\mathbf{T}_2$ . Clearly the semijoin is not commutative, because that would imply that the semijoin of  $\mathbf{T}_2$  and  $\mathbf{T}_1$ , based on the same join condition, consists of the tuples of  $\mathbf{T}_2$  that participate in the join and that the tuples of  $\mathbf{T}_1$  that participate in the join are the same as the tuples of  $\mathbf{T}_2$  that participate in the join.

More formally,  $\mathbf{T}_1 \bowtie_{\text{join-condition}} \mathbf{T}_2$  is defined to be the projection over the columns of  $\mathbf{T}_1$  of the join of  $\mathbf{T}_1$  and  $\mathbf{T}_2$ :

$$\pi_{\text{attributes}(\mathbf{T}_1)}(\mathbf{T}_1 \bowtie_{\text{join-condition}} \mathbf{T}_2)$$

Again the projection over the attributes of  $\mathbf{T}_1$  is different than the projection over the attributes of  $\mathbf{T}_2$

2. Design a query plan for the following distributed query: An application at site B wants to compute a join  $\text{STUDENT} \bowtie_{\text{Id}=\text{StudId}} \text{TRANSCRIPT}$ , where  $\text{STUDENT}(\text{Id}, \text{Major})$  is at site B and  $\text{TRANSCRIPT}(\text{StudId}, \text{CrsCode})$  is at site C. The result should be returned to B. Assume that semijoin is not used. Also assume that

- Value lengths are:
  - Id and StudId are 8 bytes long;
  - Major is 3 bytes long;
  - CrsCode is 6 bytes long.
- STUDENT has 15,000 tuples.
- 6,000 students are registered for at least one course. On the average, each student is registered for 5 courses.

**Solution:**

TRANSCRIPT has  $6,000 * 5 = 30,000$  tuples, each 14 (8+6) bytes long. Each STUDENT tuple has 11 (8+3) bytes. The join has  $8+6+3=17$  bytes in each tuple.

1. If we send STUDENT to site C, compute the join there, and then send the result to site B, we have to send 675,000 bytes ( $= 15,000 * 11 + 30,000 * 17$ ).

2. If we send TRANSCRIPT to site B and compute the join there, we have to send 420,000 bytes ( $= 30,000 * 14$ ).

Thus, alternative 2 is better.

3. Estimate the cost of computing  $\sigma_{\text{Major}='CS'}(\text{STUDENT}) \bowtie_{\text{Id}=\text{StudId}} \text{TRANSCRIPT}$  using the semijoin strategy. Use the sizes of the STUDENT and TRANSCRIPT relations and of their attributes from the previous problem. In addition, assume that 10% of the students major in CS.

Compare this with solutions that do not use the semijoin.

**Solution:**

Send the projection of  $\sigma_{\text{Major}='CS'}(\text{STUDENT})$  on Id to site C:  $8 \times 1,500 = 12,000$  bytes. Among these, the number of registered students can be estimated as  $1,500 \times 6,000 / 15,000 = 600$ . Therefore, the join at site C will have  $600 \times 5 = 3,000$  tuples. Send them to B at cost  $3,000 \times 17 = 51,000$ . The total cost is  $51,000 + 12,000 = 63,000$ .

If we use Alternative 2 in the previous problem, then the cost is still 420,000. If we use Alternative 1, we can send only the CS majors to site C:  $1,500 \times 11 = 16,500$ . Since the join was previously estimated to have 3,000 tuples, sending them to B will cost  $3,000 \times 17 = 51,000$  bytes. The total therefore is  $51,000 + 16,500 = 67,500$ . Therefore, the semijoin-based plan is better.



4. Consider the following query:

$$\pi_{\text{CustId,Price}}(\sigma_{\text{Category}='Diapers'}(\text{TRANSACTION} \bowtie_{\text{ItemId=Id}} \text{ITEM}))$$

Assume that TRANSACTION has the attributes CustId, ItemId, and Date. The ITEM relation has the attributes Id, Category, and Price, where Id is a key.

- (a) Rewrite this query using the semijoin operator  $\ltimes$  or  $\Join_{\text{semi}}$ .
- (b) Find an optimal query evaluation plan with respect to the communication cost.

Questions (a) and (b) are independent of each other (the answer in (a) does not need to be optimal with respect to the cost).

For question (b), assume that the query was issued at site A; the TRANSACTION relation is at site B with 10,000 tuples; and the ITEM relation is at site C with 1,000 tuples. Further assume that each tuple is 12 bytes long and each value of an attribute requires 4 bytes. There are 10 items in a category on the average. The selectivity factor of any particular item (i.e., the probability that it appears in any given tuple in TRANSACTION) is 0.001.

### Solution:

- (a) For example,  $\pi_{\text{CustId,Price}}((\text{TRANSACTION} \Join_{\text{semi}} \sigma_{\text{Category}='Diapers'}(\text{ITEM})) \Join_{\text{semi}} \sigma_{\text{Category}='Diapers'}(\text{ITEM}))$
- (b) i. Compute  $\pi_{\text{Id,Price}}(\sigma_{\text{Category}='Diapers'}(\text{ITEM}))$  which has the size 10 tuples \* 12 bytes = 120 bytes.  
Send the result to site B.
- ii. Perform a join at site B.  
On the average, every item occurs in  $10000 * 0.001 = 10$  tuples, so the join has 100 tuples.  
Projecting the result on CustId and Price makes each tuple 8 bytes long. Therefore, the size of the result is  $100 * 8 = 800$  bytes. This result is sent to site A.

Note that the semijoin is not optimal because after computing

$$\text{TRANSACTION} \Join_{\text{semi}} \sigma_{\text{Category}='Diapers'}(\text{ITEM})$$

we would have to send the result to either site C or A. This is pure extra cost.

## Problems for Chapter 17: OLAP and Data Mining

1. Design SQL queries for the supermarket example that will return the information needed to make a table similar to that of Figure 20.10, except that markets are aggregated by state and time is aggregated by months.

- (a) Use CUBE or ROLLUP operators.

**Solution:**

```
SELECT M.Region, T.month, SUM(S.Sales_Amt)
FROM   SALES S, MARKET M, TIME T
WHERE  S.Market_Id = M.Market_Id AND S.Time_Id = T.Time_Id
GROUP BY CUBE (M.Region, T.month)
```

- (b) Do not use CUBE or ROLLUP operators.

**Solution:**

```
SELECT M.Region, T.month, SUM(S.Sales_Amt)
FROM   SALES S, MARKET M, TIME T
WHERE  S.Market_Id = M.Market_Id AND S.Time_Id = T.Time_Id
GROUP BY M.Region, T.month
```

```
SELECT M.Region, SUM(S.Sales_Amt)
FROM   SALES S, MARKET M
WHERE  S.Market_Id = M.Market_Id
GROUP BY M.Region
```

```
SELECT T.month, SUM(S.Sales_Amt)
FROM   SALES S, TIME T
WHERE  S.Time_Id = T.Time_Id
GROUP BY T.month
```

```
SELECT SUM(S.Sales_Amt)
FROM   SALES S
```

2. Consider the following two-dimensional cube:

| SALES    |    | Time |    |    |    |
|----------|----|------|----|----|----|
| -----    |    | t1   | t2 | t3 | t4 |
| Location | NY | 3    | 5  | 2  | 3  |
|          | PA | 4    | 5  | 2  | 2  |
|          | OR | 1    | 3  | 2  | 2  |
|          | WA | 5    | 4  | 2  | 1  |

where Location and Time are dimensions represented by the following tables:

| TIME  | TimeId | Month    | LOCATION | State | Region |
|-------|--------|----------|----------|-------|--------|
| ----- |        |          | -----    |       |        |
|       | t1     | January  |          | NY    | NE     |
|       | t2     | January  |          | PA    | NE     |
|       | t3     | February |          | OR    | NW     |
|       | t4     | February |          | WA    | NW     |

Assume that the cube represents a relation with the following schema:

SALES(Time, Location, Amount)

Consider the following OLAP query:

```
SELECT T.Month, L.Region, SUM(S.Amount)
FROM SALES S, TIME T, LOCATION L
WHERE S.Time = T.TimeId AND S.Location = L.State
GROUP BY ROLLUP(T.Month, L.Region)
```

- Show the actual SQL queries that will be posed as a result of this single rollup.
- Show the cubes (of 2, 1, and 0 dimensions) produced by each of these queries.
- Show how the lower-dimension cubes are computed from higher-dimension cubes.

3. Perform the a priori algorithm on the following table to determine all two-item associations with support of at least 0.5 and confidence of at least .

| PURCHASES | Transaction_id | Product   |
|-----------|----------------|-----------|
|           | 001            | diapers   |
|           | 001            | beer      |
|           | 001            | popcorn   |
|           | 001            | bread     |
|           | 002            | diapers   |
|           | 002            | cheese    |
|           | 002            | soda      |
|           | 002            | beer      |
|           | 002            | juice     |
|           | 003            | diapers   |
|           | 003            | cold cuts |
|           | 003            | cookies   |
|           | 003            | napkins   |
|           | 004            | cereal    |
|           | 004            | beer      |
|           | 004            | cold cuts |

**Solution:** The support for the various items is as follows:

| Item      | Support      |
|-----------|--------------|
| diapers   | $3/4 = 0.75$ |
| beer      | $3/4 = 0.75$ |
| popcorn   | $1/4 = 0.25$ |
| bread     | $1/4 = 0.25$ |
| cheese    | $1/4 = 0.25$ |
| soda      | $1/4 = 0.25$ |
| juice     | $1/4 = 0.25$ |
| cold cuts | $2/4 = 0.5$  |
| cookies   | $1/4 = 0.25$ |
| napkins   | $1/4 = 0.25$ |
| cereal    | $1/4 = 0.25$ |

It follows that only the associations among diapers, beer, and cold cuts can have support of 0.5. We need to calculate such support:

| Item-pair         | Support      |
|-------------------|--------------|
| diapers,beer      | $2/4 = 0.5$  |
| diapers,cold cuts | $1/4 = 0.25$ |
| beer,cold cuts    | $1/4 = 0.25$ |

Therefore, the only associations with support of at least 0.5 can occur between beer and diapers. Both of the following associations have confidence of 0.5:

*Purchase\_diapers* => *Purchase\_beer*

*Purchase\_beer* => *Purchase\_diapers*

4. Consider the table

| TransactionId | Product |
|---------------|---------|
| 1             | milk    |
| 1             | beer    |
| 1             | diapers |
| 2             | beer    |
| 2             | diapers |
| 3             | duck    |
| 3             | yogurt  |
| 3             | milk    |
| 4             | milk    |
| 4             | beer    |
| 4             | paper   |
| 5             | diapers |
| 5             | beer    |
| 6             | diapers |
| 6             | duck    |
| 6             | paper   |

Perform the a priori algorithm to find two-item associations with support of at least 40%. Determine the confidence of each association that you found.

**Solution:** The support for the individual items are:

| Item    | Support      |
|---------|--------------|
| milk    | $3/6 = 0.5$  |
| beer    | $4/6 = 0.66$ |
| diapers | $4/6 = 0.66$ |
| duck    | $2/6 = 0.33$ |
| yogurt  | $1/6 = 0.16$ |
| paper   | $2/6 = 0.33$ |

Since we are interested for associations with support of at least 40%, each item in the association must have support of at least 40%. We have only three such items: milk, beer, and diapers, so only 3 potential associations need be considered:

| Item-pair     | Support      |
|---------------|--------------|
| milk, beer    | $2/6 = 0.33$ |
| beer, diapers | $3/6 = 0.5$  |
| milk, diapers | $1/6 = 0.16$ |

So, only the associations that involve diapers and beer have support more than 40%. It remains to compute the confidence of two possible associations:

| Association                | Confidence   |
|----------------------------|--------------|
| diapers $\Rightarrow$ beer | $3/4 = 0.75$ |
| beer $\Rightarrow$ diapers | $3/4 = 0.75$ |

## Problems for Chapter 18: ACID Properties of Transactions

1. The responsibility for producing ACID execution of transactions is shared by the system and the programmer. State which of the ACID properties are guaranteed by the system and which must be ensured by the programmer.

**Solution:**

The system guarantees that the execution will be Atomic, Isolated, and Durable. The program must ensure that the execution is Consistent.



2. Suppose a transaction consists of a sequence of SELECT statements followed by a commit statement. Consider a single one of these SELECT statements. State which of the ACID properties the execution of that SELECT statement maintains.

**Solution:**

The execution of the SELECT statement is Atomic and Isolated, but is not Durable and not necessarily Consistent.

3. A programmer is given the assignment of implementing a deposit transaction for a bank. The specification for that transaction states that the arguments for that transaction are to be the AccountId of the account and the amount of the deposit; the transaction is supposed to increment the amount of the balance in that account by the amount of the deposit. Instead the programmer implements a transaction that does absolutely nothing. It ignores its arguments, and as soon as it is called, the transaction commits. State which of the ACID properties the execution of that transaction satisfies.

**Solution:**

The execution is Atomic, Isolated, and Durable. It is "consistent" in that its execution maintains all the integrity constraints of the database, but it is not correct in that it does not change the database in a way as to satisfy its specification. Therefore we would say that it is not Consistent.

4. Suppose a distributed Internet transaction processing system stores information about your name and address at two different sites. One site is the central site at which you register and place an order. The second site is a local warehouse that sends you the items that you buy. When you go to the central site to change your address, one implementation of that interaction is that the central site initiates a change-of-address transaction and commits that transaction as soon as you press the *change address* button and it updates the local database. and then right after the commit it initiates a second transaction to update the address information at the warehouse site. (An alternative implementation would be to perform the change of address at both sites within a single transaction, but that would significantly lengthen the response time to the user who wanted to changes her address.) State which of the ACID properties the two-transaction implementation does not satisfy and what bad things might happen.

**Solution:**

Assuming that there is an integrity constraint that the address information at the two sites must be the same, that integrity constraint would not be satisfied during the period between the commits of the two transactions. Probably nothing bad will happen because the central site will not send the warehouse site any additional orders until after it sends the new address information.

There is also the atomicity issue that the second change-of-address transaction must definitely be initiated even if the central site should crash right after it commits the first transaction and before it initiates the second.

## Problems for Chapter 19: Models of Transactions

1. Assume each of the following procedures is called from within a transaction and then that procedure aborts. State whether or not the calling transaction aborts

- (a) A child in a nested transaction

**Solution:**

The calling transaction does not abort

- (b) The next transaction in a chained transaction

**Solution:**

The calling transaction does not abort

- (c) A subtransaction of a distributed transaction

**Solution:**

The calling transaction aborts

- (d) A subtransaction in a multilevel transaction

**Solution:**

The calling transaction does not abort

- (e) A method with declarative demarkation *RequiresNew*

**Solution:**

The calling transaction does not abort

- (f) A method with declarative demarkation *Required*

**Solution:**

The calling transaction aborts

- (g) A method with declarative demarkation *Mandatory*

**Solution:**

The calling transaction aborts

- (h) A method with declarative demarkation *Supports*

**Solution:**

The calling transaction aborts

2. Assume each of the following procedures is called from within a transaction, the procedure successfully completes (“commits”), and then the calling transaction aborts. State whether or not the called procedure aborts

(a) A child in a nested transaction

**Solution:**

The called procedure aborts

(b) The next transaction in a chained transaction

**Solution:**

The called procedure does not abort

(c) A subtransaction of a distributed transaction

**Solution:**

The called procedure aborts

(d) A subtransaction in a multilevel transaction

**Solution:**

The called procedure does not abort, but compensation is performed

(e) A method with declarative demarkation *RequiresNew*

**Solution:**

The called procedure does not abort

(f) A method with declarative demarkation *Required*

**Solution:**

The called procedure aborts

(g) A method with declarative demarkation *Mandatory*

**Solution:**

The called procedure aborts

(h) A method with declarative demarkation *Supports*

**Solution:**

The called procedure aborts

3. Suppose that declarative demarkation in J2EE was expanded to allow a new trans-attribute, *NestedChild* with the semantics:

If the calling method is not in a transaction, a new transaction is started

If the calling method is in a transaction, the called method executes with the same semantics as a child of a nested transaction.

Explain how this semantics differs from the semantics of each of the following trans-attribute

- (a) *Required*.

**Solution:**

For the *Required* attribute, if the called method is called from within a transaction and the called method aborts, the entire transaction is aborted. For the *NestedChild* attribute, if the called method is called from within a transaction and the called method aborts, the entire transaction does not abort.

- (b) *RequiresNew*

**Solution:**

For the *RequiresNew* attribute, if the called method is called from within a transaction, a new transaction is started. For the *NestedChild* attribute, if the called method is called from within a transaction, the called method executes within the calling transaction.

4. Explain in what way a recoverable queue is “recoverable.” In other words how does it differ from an ordinary queue on which one transaction might enqueue information about some work to be performed and a later transaction might dequeue that information and perform the specified work.

**Solution;**

- (a) A recoverable queue is atomic in that if a transaction that enqueued information on the queue aborts, the information is removed from the queue, and if a transaction that dequeues information from the queue aborts, the information is restored to the queue.
- (b) A recoverable queue is durable in that if a transaction that enqueued information on the queue commits and later the computer on which the queue is stored (or the disk on which it is stored) crashes, when it recovers from that crash, the information will be restored to the queue.

## Problems for Chapter 20: Implementing Isolation

1. For each of the following schedules, state whether or not it is serializable and/or recoverable

(a)  $w_2(x) \ r_1(x) \ w_1(y) \ commit_1 \ w_2(z) \ abort_2 \text{ or } commit_2$

**Solution:**

serializable, not recoverable

(b)  $r_2(x) \ w_1(x) \ r_1(y) \ commit_1 \ w_2(y) \ abort_2 \text{ or } commit_2$

**Solution:**

recoverable, not serializable

(c)  $w_2(x) \ r_1(x) \ r_1(y) \ commit_1 \ w_2(y) \ abort_2 \text{ or } commit_2$

**Solution:**

not serializable, not recoverable



2. For each of the following schedules, state whether or not it is serializable and/or strict.

(a)  $w_2(x) w_1(x) r_1(y) r_2(z)$

**Solution:**

serializable, but not strict

(b)  $r_1(x) r_2(y) w_1(y) w_2(x)$

**Solution:**

strict, but not serializable

(c)  $w_2(x) r_1(x) r_1(y) w_2(y)$

**Solution:**

not strict, not serializable

3. For each of the schedules in the above two problems that are serializable, draw a serialization graph

**Solution:**

The first schedule in each problem is serializable. In both cases, the serialization graph is

$$T_1 \rightarrow T_2$$

4. (a) Give an example of a schedule that is serializable and strict, but could not be produced by a two-phase concurrency control

**Solution:**

$r_1(x) \ w_2(x) \ r_1(y) \ r_2(z)$

$T_1$  gave up its lock on  $x$  before it got its lock on  $y$ .

- (b) Give an example of a schedule that could be produced by a timestamp ordered concurrency control, but not by a two-phase concurrency control

**Solution:**

$r_1(x) \ w_2(x) \ r_1(y) \ r_2(z)$

$T_1$  gave up its lock on  $x$  before it got its lock on  $y$ .

5. Assume each transaction in an application makes all its reads before it makes any of its writes. Assume that application is implemented both by an immediate-update pessimistic concurrency control and a deferred-update optimistic concurrency control. Show a schedule that would be allowed by the pessimistic control but not by the optimistic control.

**Solution:**

$r_1(x) \ r_2(y) \ w_1(x) \ commit_1 \ r_2(x) \ w_2(y) \ commit_2$

In the optimistic control, since  $T_1$ 's write phase occurred within  $T_2$ 's read phase, and  $T_1$  writes an item that  $T_1$  read, when  $T_2$  requests to commit, it would be aborted. In the pessimistic control, everything is cool.

6. Assume that a set of transaction has no **blind writes**; that is whenever a transaction performs a write on some item, it has previously read that item. Show that those transactions never have a write/write conflict.

**Solution:**

Whenever one transaction reads a particular item, no other transaction can write that item. So if the first transaction later writes that item, no other (active) transaction can have written that item (or read that item)

7. An application is to be designed using a **write once variable** type. The database operations are a write, which can be performed only once and writes a value into the variable, and a read, which can be performed only if the variable has previously been written and returns the value of the variable. Since these operations are partial, the concurrency control assumes there are four operations: `writeOK(x)`, `writeNO(x)`, `readOK(y)`, and `readNO(y)`. Design the concurrency control for these operations using the concept of backward commutativity

**Solution:**

| Requested Mode         | Granted Mode           |                        |                       |                       |
|------------------------|------------------------|------------------------|-----------------------|-----------------------|
|                        | <code>writeOK()</code> | <code>writeNO()</code> | <code>readOK()</code> | <code>readNO()</code> |
| <code>writeOK()</code> |                        |                        |                       | X                     |
| <code>writeNO()</code> | X                      |                        |                       |                       |
| <code>readOK()</code>  | X                      |                        |                       |                       |
| <code>readNO()</code>  |                        |                        |                       |                       |

Note that some conflicts cannot occur. For example, if a `writeOK()` operation has been granted and another transaction requests a write on the same item, the concurrency control will convert that into a `writeNO()` operation (not a `writeOK()` operation). Therefore there can never be a conflict between two `writeOK()` operations.

8. Suppose we have an object database that has operations `deposit(x)`, `withdrawOK(y)`, and `withdrawNO(y)`. However, the object database is, in fact, implemented by an SQL database using multilevel transactions in which the first level consists of the object operations above and the next lower level consists of SQL statements that implement these operations. There is a table called `Accounts`, with attributes, `AccountId` and `Balance`. We want the SQL statements that implement the `deposit(x)` and `withdrawOK(y)` operations to be compensatable. Give the SQL statements that implement the `deposit(x)` and `withdrawOK(y)` operations and are compensatable.

**Solution:**

The SQL operation must be one-to-one.

For the `deposit(x)` operation

```
UPDATE Accounts
SET Balance = Balance + x
WHERE AccountId = 'id'
```

For the `withdrawOK(y)` operation

```
UPDATE Accounts
SET Balance = Balance -y
WHERE AccountId = 'id'
```

Each of these SQL statements can be used to compensate for the other.

9. The conflict table for abstract operations  $r$ ,  $s$  and  $t$  is

| requested<br>mode | granted mode |   |   |
|-------------------|--------------|---|---|
|                   | r            | s | t |
| r                 |              |   | x |
| s                 |              | x | x |
| t                 | x            |   |   |

Consider the schedule  $S$  of three transactions,  $T_1$ ,  $T_2$  and  $T_3$ :

$$S : r_1(x) s_2(x) s_1(y) r_2(x) t_1(y) t_3(y) c_3 r_1(y)$$

where  $c$  indicates a commit operation and  $x$  and  $y$  are abstract objects.

- (a) Will the schedule be accepted by a pessimistic concurrency control that enforces serializability? Explain.

**Solution:**

Yes. The conflict between  $t_3(y)$  and  $r_1(y)$  is not a problem since  $T_3$  has committed.

- (b) Suppose  $t_1(y)$  is replaced in  $S$  by  $t_1(x)$ . Will the schedule be accepted by a pessimistic concurrency control that enforces serializability? Explain.

**Solution:**

No.  $t_1(x)$  conflicts with  $r_2(x)$  and a pessimistic concurrency control does not schedule an operation if it conflicts with a previous operation of a still active transaction.

- (c) Suppose  $T_1$  aborts after  $T_2$  has executed  $r_2(x)$  in the schedule

$$S' : r_1(x) s_2(x) s_1(y) r_2(x)$$

Assume that compensating operations exist for  $r$  and  $s$  and that the concurrency control grants no further operations until  $T_1$  is undone. Give the complete schedule that shows the compensation and explain why it correctly undoes  $T_1$ .

**Solution:**

$$S' : r_1(x) s_2(x) s_1(y) r_2(x) s_1^{-1}(y) r_1^{-1}(x)$$

Since  $s_2(x)$  and  $r_2(x)$  must commute with  $r_1(x)$  this schedule is equivalent to

$$s_2(x) r_2(x) r_1(x) s_1(y) s_1^{-1}(y) r_1^{-1}(x)$$



## Problems for Chapter 21: Isolation in Relational Databases

1. Suppose two transactions access a table, Table, with integer attributes, A and B, using the statements:

```
SELECT COUNT (*)  
FROM Table  
WHERE cond1
```

```
UPDATE Table  
SET A = A + 1  
WHERE cond2
```

Call the first statement *select(cond1)* and the second *update(cond2)*. For each of the following two schedules: if it is equivalent to a serial schedule, give that schedule; if not, give the reason(s) for why the schedule is not equivalent to the serial schedule T1, T2 and why it is not equivalent to the serial schedule T2, T1. (The subscript indicates the transaction.)

- (a) *select*<sub>1</sub>(A = 5) *update*<sub>2</sub>(A = 6) *update*<sub>2</sub>(A = 7) *select*<sub>1</sub>(A = 6)

**Solution**

equivalent to serial schedule T2, T1

- (b) *select*<sub>1</sub>(A = 5) *select*<sub>2</sub>(B = 5) *update*<sub>2</sub>(B = 6) *update*<sub>1</sub>(A = 6)

**Solution**

Not equivalent to T1, T2 since in the above schedule *update*<sub>2</sub> might change an A value in a row from 5 to 6 (or from 6 to 7) in which case the row will (will not) be affected by *update*<sub>1</sub>. If the order of these two operations is reversed (as in the serial schedule T1, T2) this will not happen, so the final state will be different.

Not equivalent to T2, T1 since values returned by *select*<sub>1</sub> in the two schedules will be different. *update*<sub>2</sub> might change the A value in a row from 4 to 5 (or 5 to 6) in which case *select*<sub>1</sub> will (will not) return the row in the serial schedule but not in the given schedule (will in the given schedule).

- (c) Consider the SQL statement

```
INSERT INTO Table (A, B)  
VALUES (v1, v2)
```

Specify all pairs of values (v1, v2) for which this statement commutes with *select*(A = 5).

**Solution**

All pairs (x, y) where  $x \neq 5$ .

2. Phantoms are prevented (in a system like Sybase) using a combination of granular locks and index locks. Assume that the system does locking at the level of tables and pages. Suppose transaction T1 reads a table T using a SELECT statement with predicate  $P$  and transaction T2 subsequently attempts to insert a row that satisfies  $P$ . Specify all the locks that must be acquired by each statement and state where the lock conflict occurs that prevents the phantom:

- (a) Assuming there is no useable index for the SELECT statement.

**Solution**

SELECT gets shared read lock on T. INSERT requests IX lock on T and X lock on page into which row is to be inserted. Conflict between S and IX lock on T.

- (b) Assuming there is a useable index for the SELECT statement.

**Solution**

SELECT gets IS lock on T, S lock on pages containing rows satisfying  $P$ , and S lock on leaf index pages containing pointers to pages containing rows satisfying  $P$ .

INSERT requests IX lock on T, X lock on page into which row is to be inserted, and X lock on leaf index page into which index entry for row is inserted.

Conflict between S and X lock on leaf index page.

3. The write skew anomaly was described in connection with SNAPSHOT isolation.

- (a) Give an example of a schedule of read and write operations that exhibits write skew among two transactions, T1 and T2.

**Solution**

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

- (b) Is write skew possible at SNAPSHOT isolation? Justify your answer by explaining how the algorithm that implements SNAPSHOT isolation does or does not allow the schedule of the previous part to happen.

**Solution**

Write skew is possible. Since T1 and T2 write to different variables, the first one to commit does not create a situation that will cause the abort of the second.

- (c) Is write skew possible at READ COMMITTED? At SERIALIZABLE?

**Solution**

READ COMMITTED: yes - Since no read locks are held, each transaction can acquire the lock necessary to write and then commit.;

SERIALIZABLE: no - Since read locks are long term, the schedule will cause a deadlock

4. Two transactions run concurrently and each might either commit or abort. The transactions are chosen from the following:

$T_1 : r_1(x) w_1(y)$

$T_2 : w_2(x)$

$T_3 : r_3(y) w_3(x)$

$T_4 : r_4(x) w_4(x) w_4(y)$

In each of the following cases state (yes or no) whether the resulting schedule will always be serializable and atomic. If the answer is no give a schedule that causes the violation.

- (a)  $T_1$  and  $T_2$  both running at READ UNCOMMITTED.

**Solution**

no -  $w_2(x) r_1(x) w_1(y) c_1 a_2$

- (b)  $T_2$  and  $T_3$  both running at READ UNCOMMITTED.

**Solution** yes

- (c)  $T_1$  and  $T_3$  both running at READ COMMITTED.

**Solution** yes

- (d)  $T_1$  and  $T_3$  both running at READ COMMITTED.

**Solution**

no -  $r_1(x) r_3(y) w_3(x) w_1(y) c_1 c_3$

- (e)  $T_1$  and  $T_3$  both running at snapshot isolation.

**Solution**

no -  $r_1(x) r_3(y) w_1(y) w_3(x) c_1 c_3$

- (f)  $T_1$  and  $T_4$  both running at snapshot isolation.

**Solution** yes

5. (a) What conditions must be met for two operations,  $op_1$  and  $op_2$ , to commute?

**Solution**

For all initial states the sequences  $op_1 op_2$  and  $op_2 op_1$  leave the database in the same final state and each operation returns the same result to the executing transactions

- (b) Table T has integer attributes A, B, C, and D, and is accessed by the statement

```
SELECT SUM (A)
(S1)  FROM T
      WHERE 0 <= B <= 10
```

- i. For what values of x does S1 commute with S2? Explain

```
UPDATE T
(S2)  SET A = A + 2
      WHERE B = :x
```

**Solution**

$x > 10$  or  $x < 0$ . In that case the sets of rows accessed by S1 and S2 are disjoint

- ii. For what values of x does S1 commute with S3? Explain.

```
UPDATE T
(S3)  SET A = A + 2
      WHERE C = :x
```

**Solution**

No values of x. For any value of x there is an initial state in which T contains a row satisfying  $C = x$  and  $0 \leq B \leq 10$ . The statements do not satisfy the condition for commutativity in that state.

- iii. For what values of x does S1 commute with S4? Explain.

```
UPDATE T
(S4)  SET D = D + 2
      WHERE B = :x
```

**Solution**

All values of x. Since S1 reads only A and S2 updates D the order of execution of the statements has no effect on the final state or values returned

- iv. Transaction T1 has statement S1 followed by S5 embedded in it, and transaction T2 has S2 embedded in it and these are the only SQL statements in the transactions. Assuming x is 5, can S2 be interleaved between S1 and S5 when the transactions are executed at READ COMMITTED? Can non-serializable schedules of these two transactions occur at READ COMMITTED? Explain.

```
SELECT SUM (C)
(S5)  FROM T
      WHERE 0 <= B <= 10
```

**Solution**

Yes it can be interleaved since T1 releases its read locks.

No non-serializable schedule is possible. S2 conflicts with S1 when  $x = 5$ , but there is no other conflict, so the schedule S1 S2 S5 is equivalent to the serial schedule S1 S5 S2 since S5 and S2 commute.;

- v. Answer the previous question if the isolation level is REPEATABLE READ.

**Solution**

No interleaving is possible since T1 holds long term read locks (unless no row satisfies  $B = 5$ , in which case interleaving is possible).

No nonserializable schedule is possible since no interleaving is possible.

6. Assuming there are no indices, state what S, X, IS, and IX locks are obtained when granular locking is used to implement

(a) An UPDATE statement (on a single table)

**Solution:**

An IX lock on the table and X locks on the changed tuples

(b) A SELECT statement (for a single table) at REPEATABLE READ

**Solution:**

An IS lock on the table and (long-term) S locks on the tuples that were read

(c) A SELECT statement (for a single table) at SERIALIZABLE. Explain why phantoms cannot occur.

**Solution:**

An S lock on the table. No write transaction can get an IX lock on the table.

7. Give examples of schedules in which

- (a) A read-only transaction executes at READ COMMITTED and sees an inconsistent database

**Solution:**

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

$T_2$  is transferring money from  $x$  to  $y$ .

- (b) One read/write transaction executes at READ COMMITTED and another read/write transaction executes at SERIALIZABLE and the resulting schedule exhibits a lost update.

**Solution:**

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

Both transactions are depositing money in an account.

- (c) Two read/write transactions execute at READ COMMITTED and produce a write skew.

**Solution:**

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



## Problems for Chapter 22: Atomicity and Durability

1. Figure 1 shows the final portion of the log seen by the recovery procedure on restart after a crash.

- (a) What transactions were active at the time of the crash?

**Solution**  $T_0, T_1, T_4, T_5$

- (b) What should the value of  $x$  be when recovery completes?

**Solution** 5

- (c) What is the value of  $x$  seen by  $T_6$

**Solution** 2

|  |                 |                |                                          |                |                 |                 |                                           |                |
|--|-----------------|----------------|------------------------------------------|----------------|-----------------|-----------------|-------------------------------------------|----------------|
|  | update<br>$T_1$ | begin<br>$T_2$ | update<br>$T_2$<br>$x$<br>bef 0<br>aft 2 | begin<br>$T_3$ | commit<br>$T_2$ | update<br>$T_4$ | check<br>$T_0$<br>$T_1$<br>$T_4$<br>$T_3$ | begin<br>$T_6$ |
|--|-----------------|----------------|------------------------------------------|----------------|-----------------|-----------------|-------------------------------------------|----------------|

|                                          |                |                 |                 |                                          |
|------------------------------------------|----------------|-----------------|-----------------|------------------------------------------|
| update<br>$T_6$<br>$x$<br>bef 2<br>aft 5 | begin<br>$T_5$ | commit<br>$T_3$ | commit<br>$T_6$ | update<br>$T_4$<br>$x$<br>bef 5<br>aft 6 |
|------------------------------------------|----------------|-----------------|-----------------|------------------------------------------|

Figure 2: Log on recovery from crash.

2. Figure 2 shows the final portion of the log at a particular point in time. Assume that fuzzy checkpoints are used and all update records related to items  $x$  and  $y$  and all checkpoints in this portion of the log are shown.

- (a) What value(s) might  $y$  have in the database (on non-volatile storage)? Explain.

**Solution** The page containing  $y$  might or might not have been written in the period of time over which these log records were posted since only one checkpoint record follows the update record for  $y$ . Hence the value of  $y$  in the database might be 1 or 3.

- (b) What value(s) might  $x$  have in the database (on non-volatile storage)? Explain.

**Solution** 2, 4, 6. The page containing  $x$  must have been written no later than the time the second checkpoint record was written, since it either was dirty when the first checkpoint record was written or it was written between the first update to  $x$  and the first checkpoint. Hence the value 2 is possible but the value 0 is not. It might also have been written after the second and third updates to  $x$ , in which case the value of  $x$  would be 4 or 6.

- (c) Commit records are not shown in the figure and we have assumed that all updates were made by the same transaction,  $T$ . Could your answer change if commit records had been shown or if the updates were made by multiple transactions? Explain.

**Solution** No. Since update records contain both before and after images the commit of a transaction does not force pages out of the cache and hence the arguments given in the previous two parts are unchanged.

|  |                                    |  |       |  |                                    |                                    |                                    |  |
|--|------------------------------------|--|-------|--|------------------------------------|------------------------------------|------------------------------------|--|
|  | update<br>T<br>x<br>bef 0<br>aft 2 |  | check |  |                                    | update<br>T<br>y<br>bef 1<br>aft 3 | update<br>T<br>x<br>bef 2<br>aft 4 |  |
|  |                                    |  | check |  | update<br>T<br>x<br>bef 4<br>aft 6 |                                    |                                    |  |

Figure 3: A snapshot of the log.

3. Are the following operations idempotent?

(a)  $x := |x|$

**Solution** yes

(b)  $x := 5$

**Solution** yes

(c)  $x := x + 1$

**Solution** no

(d)  $x := \lceil x \rceil$

**Solution** yes

(e)  $x := y$

**Solution** yes

4. Suppose the entire mass storage system were organized in such a way that I/O requests were handled in a FCFS fashion.

- (a) Would the write-ahead feature still be required? Explain.

**Solution:**

Yes. It is still necessary to ensure that the before image in the log record is durable before the database is updated in order to guarantee recoverability.

- (b) Would forced writes still be necessary? Explain.

**Solution:**

Assuming that FCFS means that, in addition to initiating the servicing of requests in the order of arrival, one request is serviced before the next is started, then all writes performed by the I/O system are automatically forced.

- (c) Are log sequence numbers required? Explain.

**Solution:**

Yes. The write-ahead feature requires that when a dirty page is to be flushed from the cache, update records that pertain to it have been written from the log buffer. The log sequence number makes it possible to check this condition quickly.

5. How might logging be simplified in a system that has a mirrored disk?

**Solution:**

The difference between a mirrored and a non-mirrored situation is that when a dirty page is to be flushed from the cache it is written to two independent disks rather than one. This does not change the recoverability requirements. The write-ahead feature is still required so that rollback is possible. After images are still needed for durability since it is still possible that the database has not been updated with the new values that a transaction has written when the transaction completes. Pass 1 and pass 3 of the recovery procedure are still needed to abort active transactions and pass 2 is still needed to roll the database forward.

6. Show that the two situations not handled correctly by the two-pass dump procedure are handled correctly by the three-pass dump procedure that uses two checkpoint records. List the time sequence of relevant events. Your sequence should include all events related to  $x$ , the dump and media failure, updating the log on mass store with checkpoint, commit/abort and update records, reading and updating of  $x$  in cache and mass store before media failure, and use of log records on recovery (identify the pass). Consider the following two cases:

- (a) Transaction T writes a new value to  $x$  and commits before the dump starts, but the new value is not seen by the dump.

**Solution:**

write CK1, update record for  $x$  written, T writes  $x_{new}$  to cache,  
 T commits, write CK2, start dump, dump reads  $x_{old}$  from mass store,  
 end dump, update record for  $x$  written to log,  
 $x_{new}$  flushed from cache, media fails, recovery writes  $x_{new}$  (pass 2)

- (b) Transaction T's update to  $x$  is seen by the dump, but T aborts after the dump completes. Consider two subcases: the abort record is written before the failure occurs, failure occurs before the abort record can be written.

**Solution:**

case 1 - abort record written before failure occurs:

write CK1, update record for  $x$  written, T writes  $x_{new}$  to cache,  
 $x_{new}$  flushed to database, write CK2, start dump, dump reads  $x_{new}$ ,  
 end dump, T decides abort, compensating log record for  $x$  written,  
 $x_{old}$  restored, abort record for T written, media fails,  
 recovery writes  $x_{new}$  during pass 2, recovery writes  $x_{old}$  during pass 2

case 2 - failure occurs before abort record can be written (hence, recovery sees T as an active transaction):

The case 1 sequence is modified by deleting "abort record for T written" and appending the two events:  $x_{new}$  restored during pass 3 (using the compensating log record),  $x_{old}$  restored during pass 3 (using the update record).

## Problems for Chapter 23: Architecture of Transaction Processing Systems

1. Describe some advantages of the three-tiered architecture compared with the two-tiered architecture

**Solution:**

- (a) It is more secure since the application code is not at the user site
  - (b) The application code can be more easily modified when necessary
  - (c) It is more scalable to large numbers of users
  - (d) The client machine can be smaller
2. Describe some advantages of using stored procedures.

**Solution:**

- (a) They are more secure
- (b) They can be more easily modified when necessary
- (c) They reduce the amount of communication needed
- (d) The procedure can be prepared before it is needed
- (e) Authorization can be done by the DBMS at the procedure level.
- (f) The application does not have to know the details of the database design

3. When a Web browser invokes a Web server in a client/server session, describe how the Web server maintains context information between invocations

**Solution:**

The usual approach is to use a cookie as a context handle. When the browser invokes the server, the server places in the browser a cookie containing information that can be used as a pointer to the database on the server that contains the context information about the session.

4. Describe how the application and the various resource managers communicate with the transaction manager using the X-Open interface in order to achieve global atomicity

**Solution:**

When the application wants to start a new transaction, it calls *tx\_begin*, which tells the transaction manager to start a new transaction. The transaction manager returns the transaction id.

The first time the application invokes a resource manager, the resource manager informs the transaction manager that it is part of the transaction by calling *xa\_reg*. The call includes the transaction id.

When the application wants to end a transaction, it calls either *tx\_commit* or *tx\_rollback* to tell the transaction manager to either commit or rollback the transaction. The call includes the transaction id. The transaction manager then performs some atomic commit protocol involving all the resource managers and the application.



5. When a transaction invokes a service from a resource manager using a transactional remote procedure call, state how that call is different from an ordinary remote procedure call.

**Solution:**

The transactional remote procedure call includes the tid (the transaction Id) of the calling transaction. If that is the first call to that resource manager by that transaction, the stub at the resource manager notifies the transaction manager using the xa interface before invoking the service

6. Describe how an intruder can use an event broker to eavesdrop on many aspects of an application.

**Solution:**

Since anyone can register with the event broker to be notified when a specific event occurs, the intruder can register to be notified whenever any event of interest to him occurs. All the intruder has to know is the name of the event.

7. State whether each of the following statements is true or false.

- (a) Only session beans and message-driven beans can have bean-managed transactions

**Solution:**

True

- (b) Entity beans must execute within a transaction

**Solution:**

True

- (c) With container-managed persistence, the bean programmer must declare within the entity bean, a *get* and a *set* method for each persistent field

**Solution:**

True

- (d) A message-driven bean is stateless

**Solution:**

True

- (e) A client that wants to establish a new session bean uses that bean's home interface.

**Solution:**

True

- (f) A client that wants to locate an instance of an entity bean uses that bean's home interface.

**Solution:**

True

- (g) A client that wants to access an entity or session bean on the server uses that bean's remote interface

**Solution:**

True

- (h) A client that wants to communicate asynchronously with some bean on the server uses a message-driven bean

**Solution:**

True

## Problems for Chapter 24: Implementing Distributed Transactions

1. Describe what happens in the two-phase commit protocol with presumed abort if the coordinator and one of the cohorts both crash at the same time in each of the following situations.

- (a) The cohort has not yet received a prepare message

**Solution:**

When the cohort recovers, it aborts,

When the coordinator recovers, it will not find any transaction record in its log and it does nothing. If it receives a status request from some other cohort (to which it had send a prepare message), it replies: abort.

- (b) The cohort is in its prepared state.

**Solution:**

When the cohort recovers, it sends a message to the coordinator asking the status of the transaction.

When the coordinator recovers, it retrieves the transaction record portion of the completion record from its log and stores it in volatile. Then it sends commit messages to all cohorts from which it has not yet received a done message. (It might or might not have send those cohorts a commit message before it crashed.)

- (c) The cohort has committed and sent a done message.

**Solution:**

When he cohort recovers, it does nothing since it has already committed.

When the coordinator recovers, it looks in its log. If it finds a completion record in its log, it knows it has received done messages from all cohorts, and it does nothing. If it finds a commit record but no completion record in its log, it retrieves the transaction record portion of the commit record and stores it in volatile memory. Then it sends commit messages to all cohorts from which it has not yet received a done message. (It might or might not have send those cohorts a commit message before it crashed.)

2. (a) Explain why the done message is needed for the presumed abort property of the two-phase commit protocol

**Solution:**

When the coordinator has received a done message from a cohort to which it has sent a commit message, it knows that that cohort will never again ask the status of the transaction. So when it has received done messages from all the cohorts, it knows it can delete the transaction record from its volatile memory.

- (b) Explain why a cohort at some site cannot go into its prepared state as soon as it completes what it has been asked to do.

**Solution:**

Some other cohort might be asked to execute at that same site and use some of the data that the first cohort has accessed

- (c) Suppose a nested transaction is distributed so that the parent and each child executes at a different site. Describe an atomic commit protocol that requires only one round of messages (it is a one phase commit protocol).

**Solution:**

Before each child (conditionally) commits, it goes into a prepared state (since no other child will access any data it accessed). When the parent wants to commit, it sends commit messages to all its children. Each child then sends commit messages to all of its children. etc.

3. Give an example of nonserializable schedule produced by a distributed system where each site uses a timestamped-ordered concurrency control and the coordinator uses a two-phase commit protocol.

**Solution:**

$r_1[site\ 1](x) \ w_2[site\ 1](y) \ r_1[site\ 1](y) \ r_2[site\ 2](z) \ commit_2 \ w_1[site\ 2](z) \ commit_1$

$T_1$  is before  $T_2$  at site 1 and after  $T_2$  at site 2.

4. Consider a quorum consensus protocol in which each item is stored as  $n$  replicas, and the sizes of the read and write quorums are  $p$  and  $q$  respectively. What should the value of  $p$  and  $q$  be in each of the following situations

- (a) We want the protocol to work correctly if one site should fail, and would like  $p$  to be as small as possible.

**Solution:**

$$p = 2$$

$$q = n - 1.$$

- (b) We want the protocol to work correctly if the maximum possible number of sites should fail. Also, state how many sites can fail

**Solution:**

Assuming  $n$  is even

$$p = n/2$$

$$q = n/2 + 1$$

The number of sites that can fail is

$$n/2 - 1$$

## Problems for Chapter 25: Web Services

1. A main library offers its services to branch libraries as a BPEL process. For the purposes of inter-library loan it has a portType `loanBookPT` with a synchronous operation `synchReqOp` that a branch invokes to request a book. Request and response messages, `bookReqMsg` and `bookRespMsg`, have been declared.
  - (a) Give the WSDL declaration of `loanBookPT` in terms of `bookReqMsg`, `bookRespMsg` and `.`. Assume that the target namespace of the WSDL file is `http://www.mainlib.org/wsd1/loan`, it has been assigned the prefix `lib` in the file, and that this file contains all the abstract and concrete declarations.

**Solution:**

```
<portType name='loanBookPT'>
  <operation name='synchReqOp'>
    <input message='lib.bookReqMsg' />
    <output message='lib.bookRespMsg' />
  </operation>
</portType>
```

- (b) The messages of `loanBookPT` are to be sent as RPC-style SOAP messages over HTTP using the SOAP encoding rules. Give a WSDL binding that will accomplish this.

**Solution:**

```
<binding name='loanBookRPCBinding' type='lib:loanBookPT'>
  <soap:binding style='rpc'
    transport='http://schemas.xmlsoap.org/soap/http' />
  <operation name='lib:synchReqOp'>
    <input>
      <soap:body
        use='encoded'
        namespace='http://www.mainlib.org/wsd1/loan'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding' />
    </input>
    <output>
      <soap:body
        use='encoded'
        namespace='http://www.mainlib.org/wsd1/loan'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding' />
    </output>
  </operation>
</binding>
```

- (c) Give the skeleton of a SOAP message (omit the parameter children) that results from this binding for invoking `synchReqOp`.

**Solution:**



```

<s:Envelope xmlns:s='http://www.w3.org/2003/05/soap-envelope'
  xmlns: ... >
  <s:Body>
    <n:synchReqOp xmlns:n='http://www.mainlib.org/wsd/loan'
      s:encodingStyle='http://schemas.xmlsoap.org/soap/encoding'>
      <!-- parameters go here -->
    </n:synchReqOp>
  </s:Body>
</s:Envelope>

```

- (d) Assuming that a branch library does not have to support any port types invoked by the main library, give the declaration of a partner link type that can be used for communication between the two and the partner link declarations within the BPEL processes for the main library, Main, and a branch library, Branch.

**Solution:**

```

<plnk:partnerLinkType name='branchMainPLT'>
  <plnk:role name='main'>
    <plnk:portType name='lib:loanBookPT' />
  </plnk:role>
</plnk:partnerLinkType>

```

The following declaration should appear within Main.

```

<partnerLinks>
  <partnerLink name='branchMainPL'
    partnerLinkType='lib:branchMainPLT'
    myRole='main' />
</partnerLinks>

```

The following declaration should appear within Branch.

```

<partnerLinks>
  <partnerLink name='mainBranchPL'
    partnerLinkType='lib:branchMainPLT'
    partnerRole='main' />
</partnerLinks>

```

- (e) Give the (synchronous) receive and reply statements used within Main to accept and respond to an invocation.

**Solution:**

```

<receive partnerLink='branchMainPL' portType='lib:loanBookPT'
  operation='synchReqOp' variable='reqVar' />
  <!-- body of Main goes here -->
<reply partnerLink='branchMainPL' portType='lib:loanBookPT'
  operation='synchReqOp' variable='respVar' />

```

2. Consider a new version of the library described in Exercise 1 that supports, in addition to synchronous invocation, asynchronous invocation. Discuss the new issues that this new feature introduces paying particular attention to message contents, bindings, port types, partner link types, partner links, and the way communication is handled in Main.

**Solution:**

A possible solution is as follows:

- (a) port types - Each branch will now have to support a callback port type containing a one-way operation for asynchronously receiving a response from Main. Let this be `loanBookRespPT` and let the namespace containing WSDL declarations associated with branches be assigned prefix `br` in WSDL declarations that reference it.

```
<portType name='loanBookRespPT'>
  <operation name='asynchRespOp'>
    <input message='lib.bookRespMsg' />
  </operation>
</portType>
```

Main will have to introduce a new port type containing an operation for asynchronously receiving an invocation from a branch. A new port type should be used rather than adding a new (one-way) operation to `loanBookPT` since the binding of messages for the asynchronous operation cannot be RPC (recall that a binding applies to all operations of a port type). Let this new port type be `asynchLoanBookPT`.

```
<portType name='asynchLoanBookPT'>
  <operation name='asynchReqOp'>
    <input message='lib.asynchBookReqMsg' />
  </operation>
</portType>
```

- (b) messages - The message that invokes Main asynchronously, `asynchBookReqMsg`, will have to include a new parameter that either identifies the branch doing the invocation or supplies an endpoint reference supported by the branch. In the former case, Main keeps a table associating branch identity with an invoke statement on the branch's callback port type so that the response can be sent to the appropriate branch. This solution can be made to work if the number of branches is not great and not dynamic. The latter solution is more elegant.
- (c) partner link type - A new partner link type for asynchronous communication will have to be declared, containing two roles:

```
<plnk:partnerLinkType name='asynchBranchMainPLT'>
  <plnk:role name='main'>
    <plnk:portType name='lib:asynchLoanBookPT' />
  </plnk:role>
  <plnk:role name='branch'>
    <plnk:portType name='br:loanBookRespPT' />
  </plnk:role>
</plnk:partnerLinkType>
```

- (d) partner link - Branch processes will now have to declare a partner link:

```
<partnerLinks>
  <partnerLink name='asynchBranchMainPL'
    partnerLinkType='lib:asynchBranchMainPLT'
    myRole='branch' partnerRole='main' />
</partnerLinks>
```

Main will have to declare a partner link:

```
<partnerLinks>
  <partnerLink name='asynchMainBranchPL'
    partnerLinkType='lib:asynchBranchMainPLT'
    myRole='main' partnerRole='branch' />
</partnerLinks>
```

- (e) binding - Document literal bindings are needed for `asynchLoanBookPT` and `loanBookRespPT` since a remote procedure is no longer involved.
- (f) communication in Main - Main can no longer use a receive statement to accept an invocation since it cannot know in advance which of its two port types will be invoked next. Hence a pick statement is appropriate.

```
<pick>
  <onMessage partnerLink='branchMainPL' portType='loanBookPT'
    operation='synchReqOp' variable='synchReqVar'>
    <!-- body of handler for synchronous request goes here -->
  </onMessage>
  <onMessage partnerLink='asynchMainBranchPL'
    portType='asynchLoanBookPT'
    operation='asynchReqOp' variable='asynchReqVar'>
    <!-- body of handler for asynchronous request goes here -->
  </onMessage>
</pick>
```

3. Further generalize the Library system of the previous problem by allowing Main to concurrently handle requests from branch libraries (one request from a branch at a time). Your answer should include a discussion of process creation and correlation sets.

**Solution:**

Each of the `onMessage` clauses of the pick should have a new attribute `createInstance='yes'`. In this way a new process instance is created each time a message is received by Main.

Since an interaction between a branch and Main consists of a single request message and a single reply message in both the synchronous and the asynchronous case, a correlation set is not necessary. If, however, a branch had to send multiple messages to main to complete an interaction a correlation set is needed to distinguish among the several concurrent instances of Main that might exist. In this case we might declare the following set

```
<correlationSets>
  <correlationSet name='branchCorr'
    properties='branchId' />
</correlationSets>
```

where `branchId` is a property that has been associated with a part of a request message (using a `propertyAlias`) that identifies the requesting branch. The `onMessage` clauses of the pick would now be modified. For example, the clause accepting asynchronous invocation would be

```
<onMessage partnerLink='asynchMainBranchPL'
  portType='asynchLoanBookPT'>
  operation='asynchReqOp' variable='asynchReqVar'
  createInstance='yes'>
    <correlations>
      <correlation set='branchCorr' initiate='yes' />
    </correlations>
    <!-- body of handler for asynchronous request goes here -->
  </onMessage>
```

The `initiate` attribute indicates that the value of the property `branchId` in the message is to be used to instantiate the value of the correlation set associated with the new instance of Main created when the message is received. Subsequent `receive` or `onMessage` statements that name the correlation set will only accept messages with a matching value of `branchId`.

4. Modify the solution to Exercise 1 by handling the possibility that a branch might request a non-existent book.

**Solution:**

A fault message is now declared, `reqFaultMsg` and the port type becomes

```
<portType name='loanBookPT'>
  <operation name='synchReqOp'>
    <input message='lib.bookReqMsg' />
    <output message='lib.bookRespMsg' />
    <fault name='reqFault' message='lib:reqFaultMsg' />
  </operation>
</portType>
```

The binding must be modified to describe the SOAP format of the fault message. The reply statement in Main that returns a fault is:

```
<reply partnerLink='branchMainPL' portType='lib:loanBookPT'
  operation='synchReqOp' variable='reqFaultVar'
  faultName='reqFault' />
```

and the invoke statement in the branch process is

```
<invoke partnerLink='mainBranchPL' portType='lib:loanBookPT'
  operation='synchReqOP' inputVariable='bookReqVar'
  outputVariable='bookRespVar' />
<catch faultName='lib:reqFault' faultVariable='reqFaultVar'>
  <!-- body of fault handler goes here -->
</catch>
</invoke>
```

When a fault message is received at Branch the fault `reqFault` is raised and caught by the `catch` statement nested within the `invoke`.

## Problems for Chapter 26: Security and Electronic Commerce

1. Assume all service requests made by a client are idempotent. Is an authenticator required to defend against the three attacks on the Kerberos system discussed in the text?

**Solution:**

If requests are idempotent then replays are not a problem since multiple executions of a service request achieve the same result as a single execution. Furthermore, the authenticator does not play a role in defending against the last two attacks.

2. Suppose the authenticator was simply  $K_{sess,C\&S}[WYTS]$ . Describe an attack on Kerberos security that is now possible.

**Solution:**

*I* can copy a ticket and submit it, together with arguments and a random string chosen as an authenticator, to *S*. Since the authenticator does not contain any clear text, the result of decoding it using  $K_{sess,C\&S}$  is another string that looks like a timestamp to *S*. If the timestamp falls within the lifetime of the ticket, *S* will execute the request.

3. (a) Why is it necessary to include the child `<DigestValue>` in every `<Reference>` element?

**Solution:**

If it were not present, the signature of the `<SignedInfo>` element (the value of `<SignatureValue>`) would not be dependent on the item referenced by the URI attribute of the `<Reference>` element. Hence, the contents of that item could be changed and the signature would still apply (the value of the `<SignatureValue>` element would still be correct).

- (b) Why can't the `<SignatureValue>` element be a child of the `<SignedInfo>` element?

**Solution:**

If it were, the value of `<SignatureValue>` would have to be known before it could be computed (since it is necessary in the computation).

- (c) Using an argument similar to that given in the text for the advantage of the double digest algorithm, give an example of an attack on the `<Transforms>` element that could be used if that element were not digested as a part of the second signature.

**Solution:**

If the `<Transforms>` element were not included in the second signature, an intruder could change its value. Suppose the element originally referred to an algorithm that transformed the data by deleting the first element with tag `x`. If the signature were attached to data that had one such element then the signature would be computed over all other elements in the data. If the intruder could change the value of the `<Transforms>` element so that it referred to an algorithm that transformed the data by deleting all elements with tag `x` it could attach the `<SignedInfo>` element to a new data item that appended to the original data item a new element with tag `x`.

4. Give a SOAP message in which the entire element that is a child of Body is encrypted using a triple DES algorithm, and the (symmetric) key used to encrypt the data is sent in encrypted form in the header. The key is encrypted with a public key algorithm. The public key to be used to decrypt the symmetric key is identified to the receiver using the name “myKey” supplied with the encrypted symmetric key. Finally, <EncryptedData> is signed to identify the sender. The sender includes a certificate which the receiver can use to verify the signature.

**Solution:**

```
<s:Envelope xmlns:s='...' >
  <s:Header>
    <wsse:Security xmlns:wsse='...'
      xmlns:wsu='...'>
      <xenc:EncryptedKey xmlns:xenc='...'>
        to encrypt the message body
        <xenc:EncryptionMethod
          Algorithm='http://www.w3.org/2001/04/xmenc#rsa-1_5' />
        <ds:KeyInfo xmlns:ds='...'>
          <wsse:KeyName>
            myKey
          </wsse:KeyName>
        </ds:KeyInfo>
        <xenc:CipherData>
          <xenc:CipherValue> ... </xenc:CipherValue>
        </xenc:CipherData>
        <xenc:ReferenceList>
          <xenc:DataReference URI='#bodyId' />
        </xenc:ReferenceList>
      </xenc:EncryptedKey>
      <ds:Signature xmlns:ds='...'>
        signature of encrypted data
        <ds:SignedInfo>
          <ds:SignatureMethod
            Algorithm='http://www.w3.org/2000/09/xmldsig#dsa-sha1'>
          <ds:Reference URI='#bodyId'>
            <ds:DigestMethod Algorithm='...' />
            <ds:DigestValue> ... </ds:DigestValue>
          </ds:Reference>
        </ds:SignedInfo>
        <ds:SignatureValue> ... </ds:SignatureValue>
      </ds:Signature>
      <wsse:BinarySecurityToken xmlns:wsse='...'
        ValueType='...'
        wsu:Id='...'>
        binary certificate for signature goes here
      </wsse:BinarySecurityToken>
```



```

        <wsse:Security>
</s:Header>
<s:Body>
    <xenc:EncryptedData Id='bodyId'
        xmlns:xenc='...'
        Type='http://www.w3.org/2001/04/xmlenc#Element'>
        <xencEncryptionMethod
            Algorithm='http://www.w3.org/2001/04/xmlenc#tripleDES-cbc' />
        <CipherData>
            <CipherValue> ... </CipherValue>
        </CipherData>
        </xenc:EncryptedData>
    </s:Body>
</s:Envelope>

```

## Problems for Appendix A: 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:**  $r_1(x) \ w_2(x) \ r_2(y) \ w_2(y) \ Commit_1 \ Commit_2$

- (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 decrease 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.