Databases and Transaction Processing: An Application-Oriented Approach (Complete Version)

Test Bank: Additional Problems and Solutions

Michael Kifer, Arthur Bernstein, Philip M. Lewis

Contents

Problems for Chapter 2: The Big Picture

1. Consider a table with the columns as shown below:

   Book(ISBN,Title,Publisher,PublicationDate)

   Write the following query using SQL: *Find the titles of all books that were published on 9/9/1991*.

   Solution:

```
SELECT  Title
FROM    Book B
WHERE B.PublicationDate = '9-9-1991'
```

2. Consider a table with the columns as shown below:

   CAR(Make,ModelNum,MName,Kind)

   Write the following query using SQL: *Find the kinds of all cars whose make is "Ford"*.

   Solution:

```
SELECT C.Kind
FROM CAR C
WHERE C.Make='Ford'
```

3. Consider a table with the columns as shown below:

   MOVIE(Title,ReleaseDate,Director,StudioName)

   Express the following queries using SQL:

   (a) Find all movies released on 1/11/2002.

   (b) Find all movie titles produced by Sony Pictures.

   (c) Count the number of movies that were released on 1/11/2002.

   Solution:

```
SELECT * FROM Movies
WHERE ReleaseDate = "1/11/2002"

SELECT Title  FROM Movies
WHERE StudioName = "SonyPictures"

SELECT COUNT(*)  FROM Movies
WHERE ReleaseDate = "1/11/2002"
```

Problems for Chapter 3: The Relational Data Model

1. (a) Design a schema for a library system containing the following data.

   i. the name, unique Id and number of books on loan for each patron

   ii. the unique isbn number, title, author (each book has a single author), current status (possible values are on-shelf or on-loan), borrower Id (if book is on-loan), and shelf-Id of each book

   iii. the unique shelf-Id and capacity (number of books) of each shelf

   Show all primary and foreign keys.

   Solution:

```
CREATE TABLE PATRONS (
    Id INTEGER,
    Name CHAR(20),
    NumBorrowed INTEGER,
    PRIMARY KEY (Id) )

CREATE TABLE BOOKS (
    Isbn CHAR(20),
    Title CHAR(40),
    Author CHAR(20),
    Status CHAR(5),
    BorrowerId INTEGER,
    ShelfId INTEGER,
    CHECK (Status IN ('on-loan', 'on-shelf') )
    FOREIGN KEY (ShelfId)
            REFERENCES SHELVES (ShelfId)
    FOREIGN KEY (BorrowerId)
            REFERENCES PATRONS (Id) )

CREATE TABLE SHELVES (
    ShelfId INTEGER,
    Capacity INTEGER
    PRIMARY KEY (ShelfId) )
```

   (b) Add a constraint that enforces the restriction that a patron cannot borrow more than 5 books at a time.

   Solution:

   Add the following to PATRONS

```
CHECK (NumBorrowed < 5)
```

   (c) Add a constraint to the schema that enforces the restriction that the number of books assigned to a shelf cannot exceed the shelf capacity.

   Solution:

```
CREATE ASSERTION ShelfCapacity
    CHECK ( NOT EXISTS (
                    SELECT *
                    FROM SHELVES S
                    WHERE S.Capacity <
                        ( SELECT COUNT (*)
                          FROM BOOKS B
                          WHERE B.ShelfId = S.ShelfId) ) )
```

2. (a) Design a schema for a part of a package delivery company, which contains information about packages (PkgId, AddresseeId), addressees (Id, Name, StreetNumber, StreetName, City), and streets (StreetName, City, MinHouseNumber, MaxHouseNumber). Show the primary and foreign keys. Indicate the NOT NULL constraint wherever applicable.

Solution:

```
CREATE TABLE Package (
  PkgId INTEGER,
  AddresseeId CHAR(20) NOT NULL,
   PRIMARY KEY (PkgId)
   FOREIGN KEY (AddresseeId) REFERENCES Addressee (Id)
)

CREATE TABLE Addressee (
  Id CHAR(20),
  Name CHAR(20) NOT NULL,
  StreetNumber INTEGER NOT NULL,
  StreetName CHAR(40) NOT NULL,
  City CHAR(20) NOT NULL,
   PRIMARY KEY (Id)
   FOREIGN KEY (StreetName,City) REFERENCES Streets
)

CREATE TABLE Streets (
  StreetName CHAR(40),
  City CHAR(20),
  MinHouseNumber INTEGER NOT NULL,
  MaxHouseNumber INTEGER NOT NULL,
   PRIMARY KEY (StreetName, City)
)
```

(b) Express the constraint that the street number in the addressee's address must be within the range valid for the corresponding street.

Solution:

```
CREATE ASSERTION  ValidateRanges
CHECK ( NOT EXISTS
          ( SELECT *
            FROM  Addressee A,  Streets S
            WHERE A.StreetName = S.StreetName  AND
                  A.City = S.City  AND
                  (A.StreetNumber < S.MinHouseNumber
                     OR A.StreetNumber > S.MaxHouseNumber) )
      )
```

3. (a) Design a schema for an airline containing the following information.

   i. the unique Id and name of each passenger
   ii. the unique Id, type (possible value A, B, and C) and capacity of each plane
   iii. the unique Id, date, plane Id, starting location and destination of each flight (a plane cannot be scheduled for more than one flight on any particular day)
   iv. each reservation made by a passenger on a flight

   Show all primary and foreign keys.

   Solution:

```
CREATE TABLE Passengers (
    Id  INTEGER,
    Name  CHAR(20),
     PRIMARY KEY (Id) )

CREATE TABLE Planes (
    Id  INTEGER,
    Capacity  INTEGER,
    Type  CHAR(1),
     CHECK (Type IN ('A', 'B', 'C') )
     PRIMARY KEY (Id)  )

CREATE TABLE  Flights (
    Id  INTEGER,
    PlaneId INTEGER,
    Date  DATE,
    Start  TIME,
    Destination  CHAR(20),
     PRIMARY KEY (Id),
     UNIQUE (PlaneId, Date),
     FOREIGN KEY (PlaneId)
            REFERENCES Planes (Id)  )

CREATE TABLE Reservations (
    PassengerId  INTEGER,
    FlightId  INTEGER,
     PRIMARY KEY (PassengerId, FlightId),
     FOREIGN KEY (PassengerId)
            REFERENCES Travellers (Id),
     FOREIGN KEY (FlightId)
            REFERENCES Flights (Id) )
```

   (b) Add a constraint that enforces the restriction that the number of passengers on a plane cannot exceed the plane's capacity

   Solution:

```
CREATE ASSERTION PlaneCapacity
    CHECK ( NOT EXISTS (
        SELECT *
        FROM PLANES P, FLIGHTS F
        WHERE P.Id = F.PlaneId AND P.Capacity <
                        ( SELECT COUNT (*)
                          FROM RESERVATIONS R
                          WHERE R.FlightId = F.Id) ) )
```

(c) Add a view that shows for each reservation on '6/8/2004' the passenger's name, the starting location, and the destination.

Solution:

```
CREATE VIEW TRIPS (PassName, Start, Destination) AS
    SELECT P.Name, F.Start, F.Destination
    FROM PASSENGER P, FLIGHTS F, RESERVATIONS R
    WHERE P.Id = R.PassengerId AND R.FlightId = F.Id
                AND F.Date = '6/8/2004'
```

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

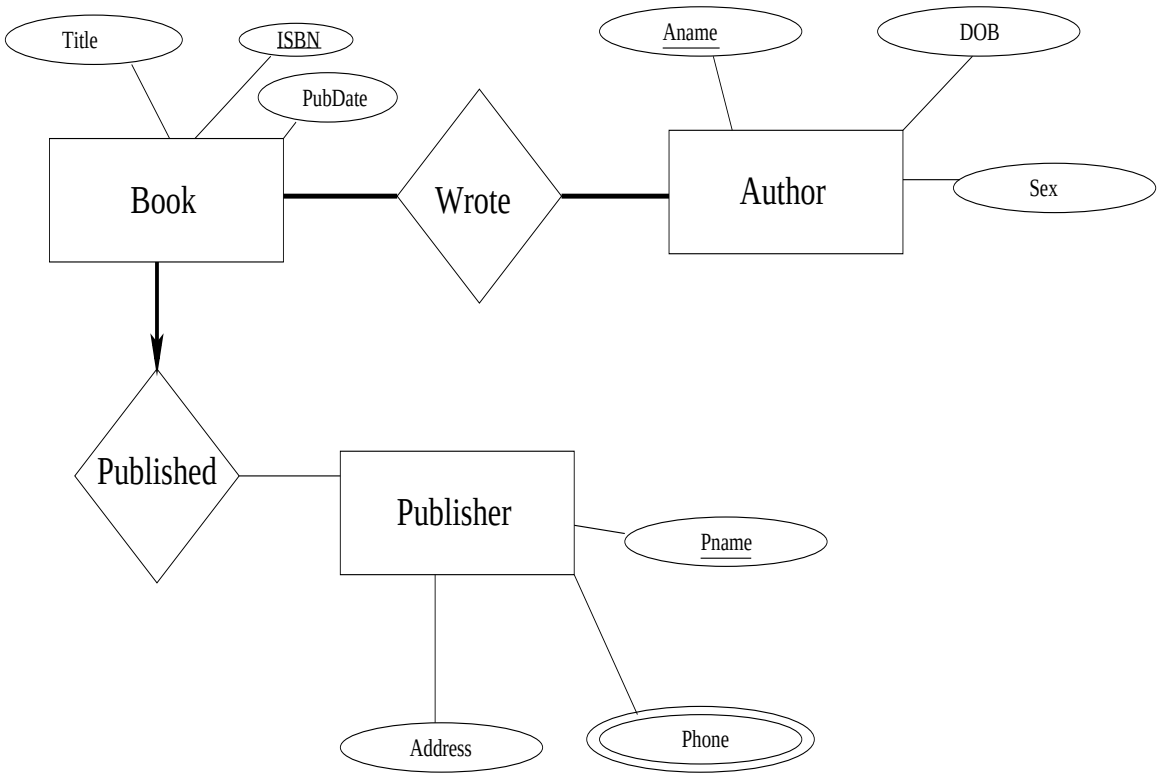
1. Consider the following enterprise, which includes books, authors and publishers. Authors are people with normal attributes, like name, date of birth, etc., but in addition they wrote one or more books. A book has the usual attributes, such as title, ISBN, publication date, etc. Publishers are companies that publish books. They have an address, phone numbers (typically more than one), name, etc.

   A book can be written by more than one author, but it can be published by only one publisher. Books do not write themselves and do not publish themselves (hint: these are constraints). An author can write more than one book and to be called an author one, of course, has to write at least one book.

   (a) Represent the above as an E-R diagram; include all relevant constraints.

   Solution:

   Assumptions: the author name is unique and the publisher name is unique.
   E_R diagram:



(b) Translate the above diagram into the relational model by supplying the appropriate CREATE TABLE statements. Note that ISBN is a 10-digit string (which can have leading zeros), sex can have only two values, 'M' or 'F', and a phone number is a 10 digit number that never starts with a zero. Specify these as domains.

Specify all the key and foreign key constraints. Try to preserve as many participation constraints as possible. List all the participation constraints that are present in the E-R diagram, but *not* in its translation to SQL.

Solution:

```
CREATE DOMAIN ISBN_TYPE CHAR(10)
CHECK( VALUE BETWEEN '0000000000' AND '9999999999').

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

CREATE DOMAIN PHONE_TYPE INTEGER
CHECK ( VALUE > 999999999 AND VALUE < 10000000000 )

CREATE TABLE Book (
ISBN ISBN_TYPE,
Title CHAR(60),
PublicationDate DATE,
PName CHAR(60) NOT NULL,
PRIMARY KEY (ISBN),
FOREIGN KEY (PName) REFERENCES Publisher
)
CREATE TABLE Author (
AName CHAR(60),
DOB DATE,
Sex SEX_TYPE,
PRIMARY KEY (AName)
)
CREATE TABLE Publisher (
PName CHAR(60),
Address CHAR(60),
PRIMARY KEY (PName)
)
CREATE TABLE Publisher_Phone (
PName CHAR(60),
Phone PHONE_TYPE,
PRIMARY KEY (PName, Phone),
FOREIGN KEY (PName) REFERENCES Publisher
)
CREATE TABLE Wrote (
ISBN ISBN_TYPE,
```

```
AName  CHAR(60),
 PRIMARY KEY (ISBN, AName),
 FOREIGN KEY (ISBN) REFERENCES Books,
 FOREIGN KEY (AName) REFERENCES Author
)
```

Participation constraints not present in SQL:

i. Author has to write at least one book.

ii. Book has to be written by at least one author.

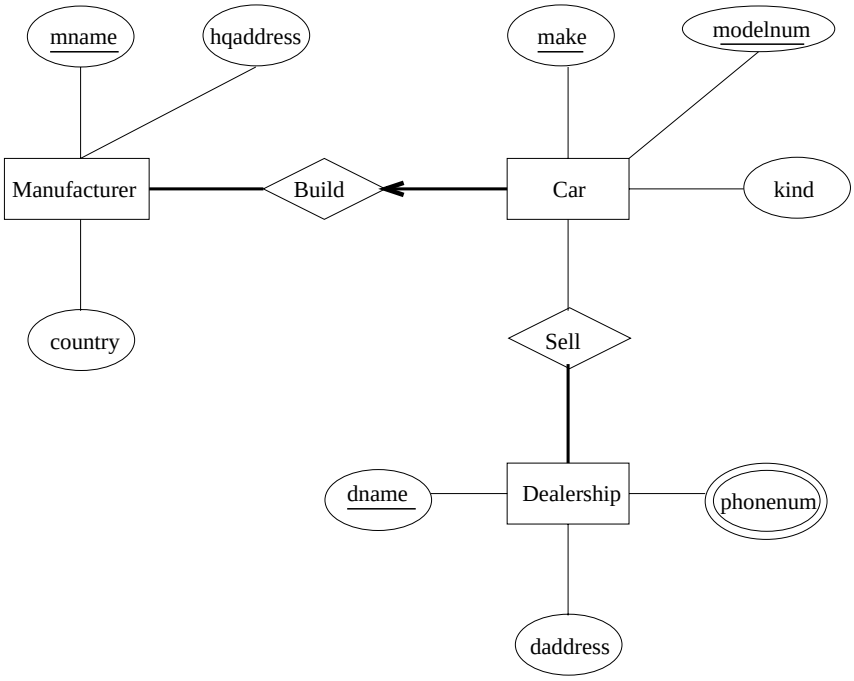
Note that we could have combined `Publisher` and `Publisher_Phone` in one table (by just adding the attribute `phone`). But then the referential integrity constraint that `Book.pname` references `Publisher.pname` will no longer be a foreign key constraint in `Book`, because `pname` will no longer be a key of `Publisher` (the key will be `(pname, phone)`). So, we would have to use an assertion to express this more general inclusion dependency.

2. Consider the following automotive enterprises, involving cars, car manufacturers, and car dealerships. Car Manufacturers are companies that build cars and they have attributes such as name, headquarters address, country of incorporation, etc. Cars have attributes such as make (e.g., Ford, Volkswagen, Chrysler), model number, kind of car (e.g., sedan, SUV, wagon), etc. A dealership sells cars. It has a name, address, and telephone numbers (typically more than one.) A manufacturer may make several different kinds and models of cars (as, for example, the manufacturer Mercedes/Chrysler does). A car is made by a single manufacturer. A dealer can sell cars from several different manufacturers but does not have to sell all the cars from a single manufacturer. For example, Nardy Honda-Pontiac in Smithtown sells Pontiacs from manufacturer GM and Hondas from manufacturer Honda, but it doesn't sell Chevrolets that are also made by GM. A manufacturer can make a car (such as a racing car) that is not sold through dealerships.

   (a) Represent the above as an E-R diagram; include all relevant constraints.

   Solution:

   Assumptions: the name of dealership is unique; the name of manufacturer is unique.
   E_R diagram:



(b) Translate the above diagram into the relational model by supplying the appropriate CREATE TABLE statements. A phone number is a 10 digit number that never starts with a zero. A kind of car is one of the following: 2-Door sedan, 4-door sedan, convertable, SUV, sports car, wagon, hatchback, light truck, or other. Specify these as domains.

Specify all the key and foreign key constraints. Try to preserve as many participation constraints as possible. List all the participation constraints that are present in the E-R diagram, but *not* in its translation to SQL.

Solution:

```
CREATE DOMAIN PHONENUM INTEGER
CHECK ( VALUE $>$ 999999999 AND VALUE $<$10000000000)

CREATE DOMAIN CARKIND CHAR(15)
CHECK ( VALUE IN
('2-door sedan','4-door sedan', 'convertable', 'SUV', 'sports car',
'wagon', 'hatchback', 'light truck', 'other'))

CREATE TABLE MANUFACTURER (
MName CHAR(30),
HQAddress CHAR(100),
Country CHAR(30),
PRIMARY KEY (MName))

CREATE TABLE CAR (
Make CHAR(20),
ModelNum INTEGER,
MName CHAR(30),
Kind CARKIND,
PRIMARY KEY (Make, ModelNum),
FOREIGN KEY (MName) REFERENCES MANUFACTURER)

CREATE TABLE DEALERSHIP (
DName CHAR(30),
DAddress CHAR(100),
PRIMARY KEY (DName))

CREATE TABLE HASPHONENUM (
DName CHAR(30),
PhoneNum PHONENUM,
PRIMARY KEY (DName, PhoneNum),
FOREIGN KEY (DName) REFERENCES DEALERSHIP ))

CREATE TABLE SELL (
DName CHAR(30),
Make CHAR(20),
```

```
ModelNum INTEGER,
 PRIMARY KEY (DName, Make, ModelNum),
 FOREIGN KEY (DName) REFERENCES DEALERSHIP,
 FOREIGN KEY (Make, ModelNum) REFERENCES CAR)
```

Other participation constraints:

i. Each manufacturer builds at least one car

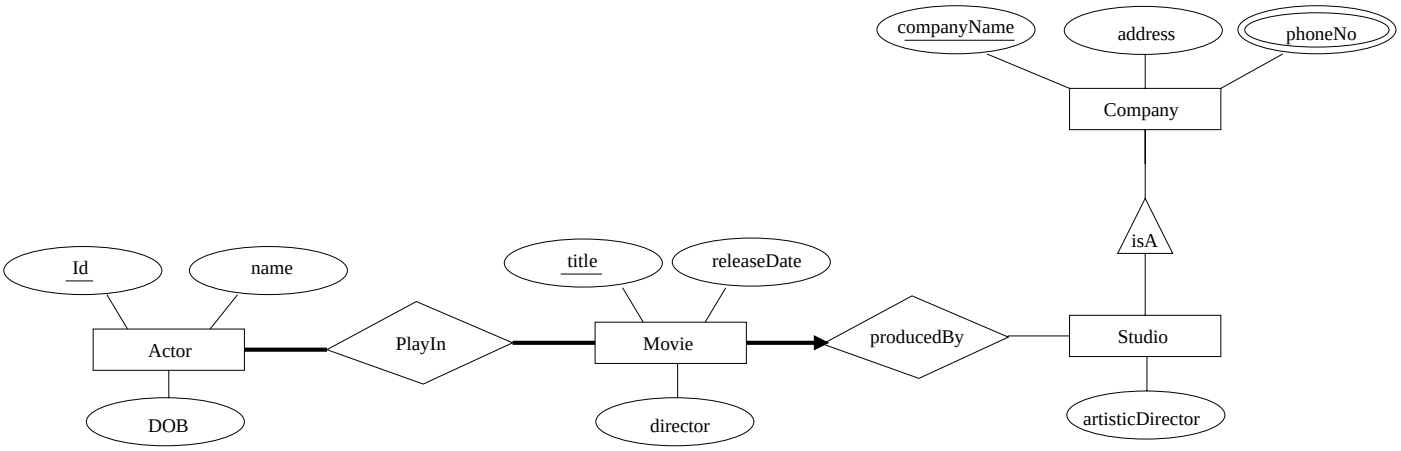
ii. Each dealership sells at least one car

3. Consider the following enterprise, which includes movies, actors, and studios that produce movies. Actors are people with normal attributes, like Id, name, date of birth, etc. Actors play in movies. A movie has the usual attributes: title, release date, director, etc. (you choose – we do not need many). Studios are companies. A company has an address, phone numbers (typically more than one), name, etc. Studios have additional attributes, such as the artistic director.

   Constraints: A movie has at least one actor, and exactly one studio makes each particular movie. Every actor played in at least one movie. Some studios may be brand new and had no time to make any movies yet.

   (a) Represent the above as an E-R diagram; include all relevant constraints.

   Solution:



(b) Translate the above diagram into the relational model by supplying the appropriate CREATE TABLE statements. Note that actor's Id is a 10-digit string and a phone number is a 10 digit number that never starts with a zero. Specify these as domains in SQL.

Specify all the key and foreign key constraints. Try to preserve as many participation constraints as possible. List all the participation constraints that are present in the E-R diagram, but *not* in its translation to SQL.

Solution:

```
CREATE DOMAIN PhoneDomain INTEGER
    CHECK (1000000000 < VALUE AND VALUE < 9999999999)

CREATE DOMAIN IdDomain CHAR(10)
    CHECK ("1000000000" < VALUE AND VALUE < "9999999999")

CREATE TABLE ACTOR (
    Id IdDomain,
    Name CHAR(20),
    DOB Date,
    PRIMARY KEY (Id)
)
CREATE TABLE MOVIE (
    Title: CHAR(20),
    ReleaseDate: Date,
    Director CHAR(20),
    StudioName CHAR(20),
    PRIMARY KEY (Title),
    FOREIGN KEY (StudioName) REFERENCES STUDIO(Name),
)
CREATE TABLE PLAYIN(
    ActorId idDomain,
    MovieTitle CHAR(20),
    PRIMARY KEY (actorId, MovieTitle),
    FOREIGN KEY (actorId) REFERENCES ACTOR(Id)
    FOREIGN KEY (MovieTitle) REFERENCES MOVIE(Title)
)
CREATE TABLE COMPANY(
    CompanyName CHAR(20),
    Phone PhoneDomain,
    Address CHAR(20),
    PRIMARY KEY (CompanyName, Phone)
)
CREATE TABLE STUDIO(
    Name: CHAR(20),
    ArtisticDirector: CHAR(20),
    PRIMARY KEY (Name)
```

```
)

-- Expresses the inclusion dependency that Studio is a Company
CREATE ASSERTION     StudioIsAcompany
  CHECK (
     NOT EXISTS ( SELECT  *
                        FROM Studio S
                        WHERE S.Name  NOT IN
                                    ( SELECT C.CompanyName
                                      FROM Company C ) )
  )
```

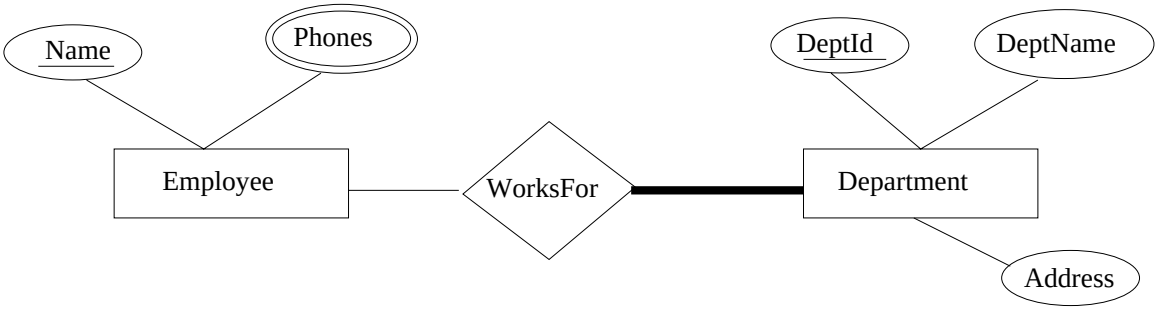
Participation constraints presented in the ER diagram, but not in its translation to SQL:

i. a movie has at least one actor

ii. every actor played in at least one movie

iii. Note: we cannot use something like FOREIGN KEY (Name) REFERENCES Company(CompanyName) in Studio to express the ISA relationship between Studio and Company. Unfortunately, CompanyName is not a key in Company, so we cannot use foreign keys here and must use assertions instead.

4. Consider the E-R diagram depicted in the figure. Write down the corresponding relational schema using SQL. Include all applicable constraints.



Solution:

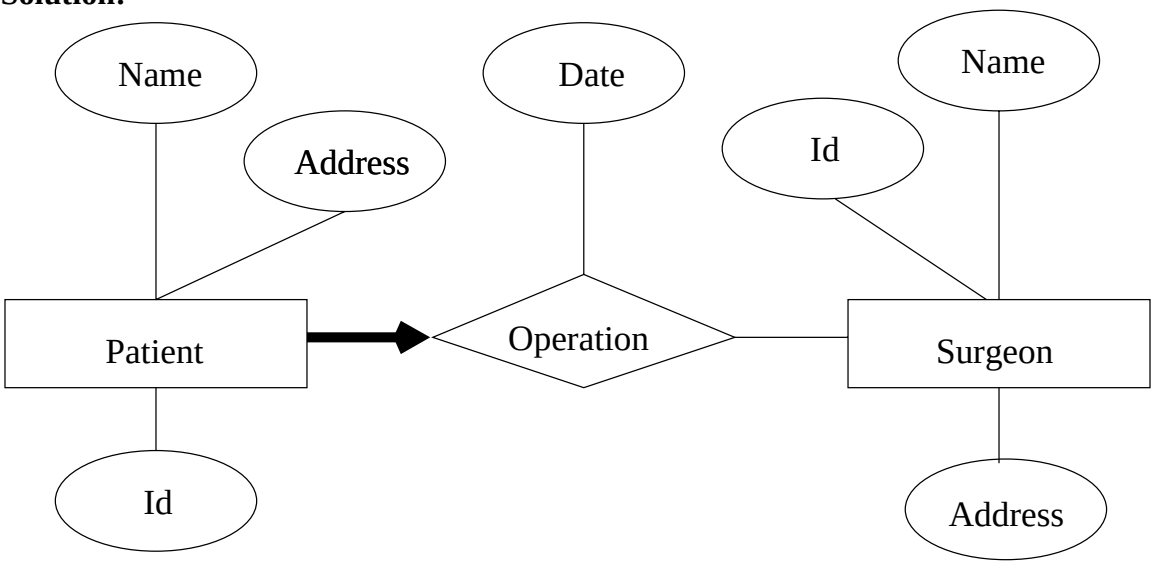
```
CREATE TABLE Department (
  DeptId  CHAR(6),
  DeptName CHAR(30)  NOT NULL,
  Address CHAR(50),
   PRIMARY KEY DeptId  )

CREATE TABLE Employee (
  Name CHAR(20),
  Phone CHAR(10),
   PRIMARY KEY (Name, Phone) )

CREATE TABLE WorksFor (
  EName CHAR(20),
  DeptId CHAR(6),
   FOREIGN KEY (EName) REFERENCES Employee(Name),
   FOREIGN KEY (DeptId) REFERENCES Department(DeptId)
```

The participation constraint cannot be represented using foreign keys, since `DeptId` is not a key in WorksFor. It can be represented as an assertion, however, which states that every `DeptId` in Department occurs somewhere in WorksFor:

```
CREATE ASSERTION DeptMustHaveEmployees
CHECK (
   NOT EXISTS ( SELECT D.DeptId
                FROM Department D
                WHERE
                     D.DeptId  NOT IN ( SELECT W.DeptId
                                        FROM WorksFor W) )
         )
```

5. (a) A hospital employs surgeons who operate on patients. Patients and surgeons are described by (unique) id's (integers), names and addresses. Each patient undergoes a single operation that is scheduled on a particular day (of type DATE) and performed by a single surgeon. A surgeon can only perform one operation a day.

   i. Give an ER diagram that describes this enterprise.
   Solution:



   ii. Translate your diagram into a schema using SQL. Include all primary and foreign key constraints.
   Solution:

```
CREATE TABLE Patient (
   Id   INTEGER,
   Name  CHAR(20),
   Address  CHAR (100),
  PRIMARY KEY Id )

CREATE TABLE Surgeon (
   Id  INTEGER,
   Name  CHAR(20),
   Address  CHAR(100),
  PRIMARY KEY Id )

CREATE TABLE Operation (
   Pid  INTEGER,
   Sid  INTEGER,
   Date  DATE,
    PRIMARY KEY Pid,
    UNIQUE (Sid, Date),
    FOREIGN KEY (Pid)  REFERENCES  Patient(Id),
    FOREIGN KEY (Sid)  REFERENCES  Surgeon(Id))
```

```
CREATE ASSERTION FULLPATIENTPARTICIPATION
    CHECK (NOT EXISTS(SELECT P.Id FROM Patient P
        EXCEPT
    SELECT O.Id FROM OPERATION O ))
```

The assertion could be replaced by the following constraint.

```
FOREIGN KEY Id REFERENCES OPERATION Pid
```

(b) How does your ER diagram and schema change if a patient can undergo several surgeries (on different days, with possibly different surgeons) and if several surgeons can be involved in a single operation? (Indicate only the changes from part 5a)

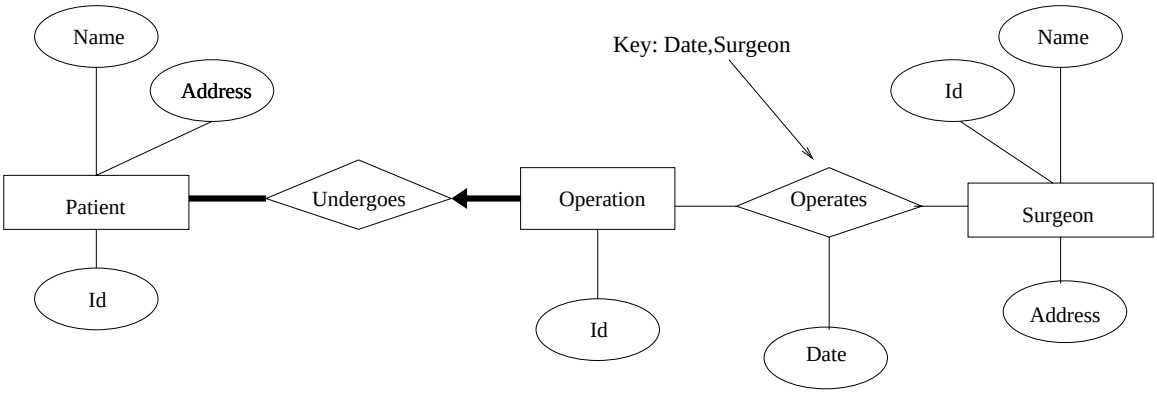
Solution:

Change to E-R diagram: dark arrow from patient to operation replaced by dark line. Although, the name of the relationship should now be OPERATES and it now does not represent a single operation, but rather who operates on whom. Change to schema:

```
CREATE TABLE OPERATES (
        Pid INTEGER,
        Sid INTEGER,
        Date DATE,
         PRIMARY KEY (Sid, Date),
         FOREIGN KEY (Pid) REFERENCES PATIENT(Id),
         FOREIGN KEY (Sid) REFERENCES SURGEON(Id))
```

Note: Adding constraint UNIQUE (Pid, Date) in an attempt to limit a patient to one surgery a day will also eliminate the possibility of multiple surgeons performing a single operation, so it is not included. In fact, since a tuple <p,s,d> means that s is one of the surgeons who operated on p on date d, the design requirement that a person can have at most one surgery per day is already taken care of.

Another solution is based on the following diagram:



Then use the standard translation into SQL.

(c) Write an assertion that ensures total participation for all surgeons.

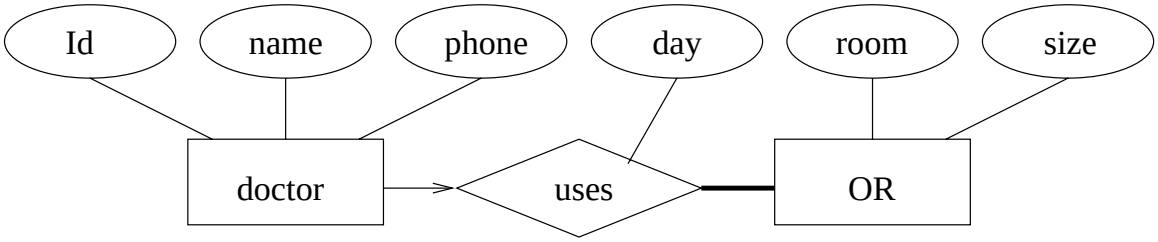
Solution:

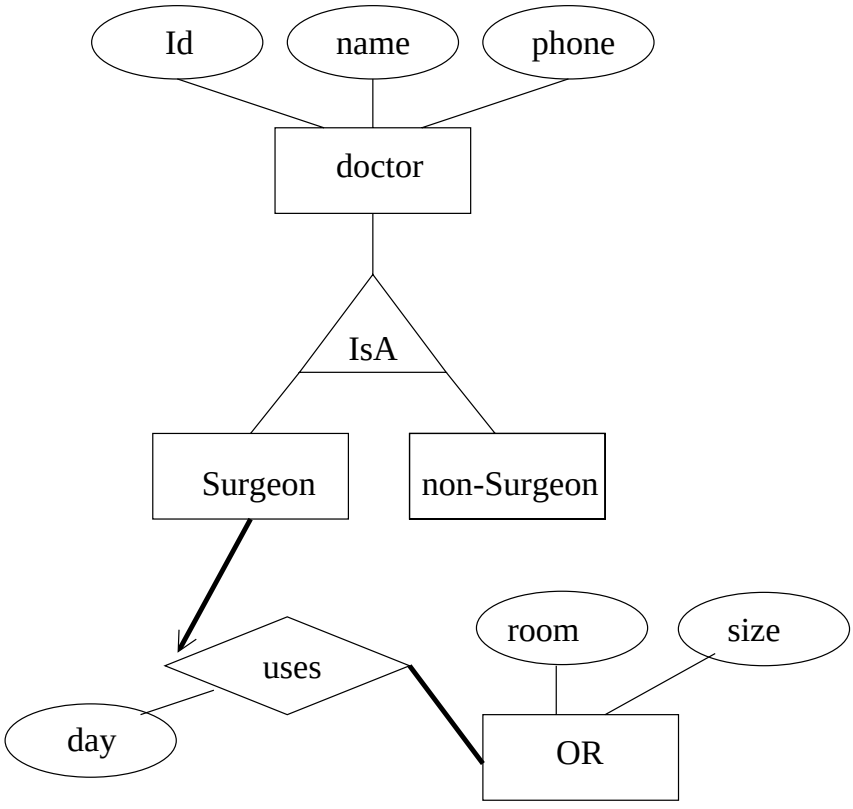
```
CREATE ASSERTION TOTAL
    CHECK NOT EXISTS (SELECT * FROM Surgeons S
                      WHERE S.Id NOT IN (
                          SELECT O.Sid FROM OPERATION O))
```

6. In a hospital each surgeon is assigned to a particular operating room (OR) on the same day each week (e.g., Dr. Smith is in OR22 every Wednesday). A surgeon gets exactly one such assignment and each OR gets some use. Other doctors (not surgeons) don't have ORs assigned to them. ORs are described by their (unique) room number and size. Each doctor is described by a (unique) Id, name, and phone number. Draw an E-R diagram (neatly) that describes this enterprise and represents all these facts.

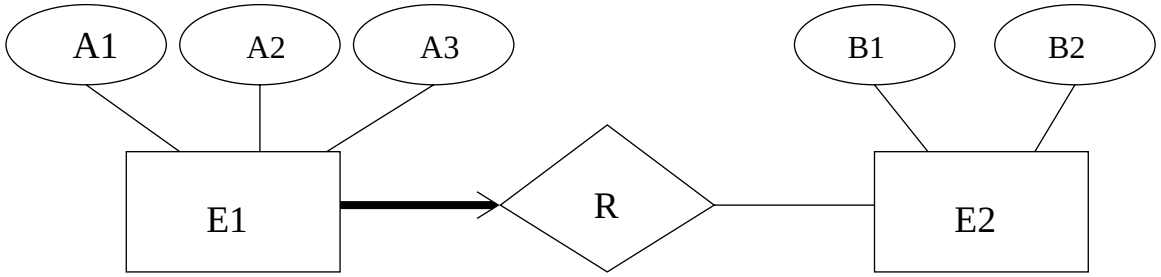
Solution:

See figure for two possible solutions.





7. (a) Translate the E-R diagram below into an SQL schema involving three tables: E1, R, and E2. Include all primary and foreign key constraints and assume all attributes have type integer.



Solution:

```
CREATE  TABLE E1 (
   A1 :  INTEGER,
    A2 :  INTEGER,
    A3 : INTEGER,
    PRIMARY KEY (A1, A2),
    FOREIGN KEY (A1, A2) REFERENCES R (A1, A2))

CREATE TABLE R (
     A1 : INTEGER,
    A2 : INTEGER,
    B1 : INTEGER,
    PRIMARY KEY (A1, A2),
         FOREIGN KEY (A1, A2) REFERENCES E1 (A1, A2),
         FOREIGN KEY (B1) REFERENCES E2 (B1))

CREATE TABLE E2 (
    B1 : INTEGER,
         B2 : INTEGER,
         PRIMARY KEY B1)
```

Alternate solution uses table E2 as above and replaces E1 and R with:

```
CREATE TABLE E1-R (
         A1: INTEGER,
    A2: INTEGER,
         A3: INTEGER
    B1 : INTEGER,
    PRIMARY KEY (A1, A2),
         FOREIGN KEY (B1) REFERENCES E2 (B1))
```

(b) Add the following constraint to the above schema. Don't rewrite the whole schema; just provide the material to be added and tell me where it goes.

i. In any row of E1, attribute A3 must be less than A2.

Solution:

Add constraint CHECK A3 ¡ A2 in CREATE TABLE E1

ii. The number of rows in R is less than any value of B2

Solution:

Add the following assertion to the schema

```
CREATE ASSERTION X
    CHECK (SELECT COUNT (*) FROM R ) <
        ( SELECT MIN (B2) FROM E2 )
```

(c) Create a view of this database that provides the values of A3 and B1 for all entities related by R.

Solution:

```
CREATE VIEW X (A3, B1) AS
    SELECT E1.A3, R.B1
    FROM E1, R
    WHERE E1.A1 = R.A1 AND E1.A2 = R.A2
```

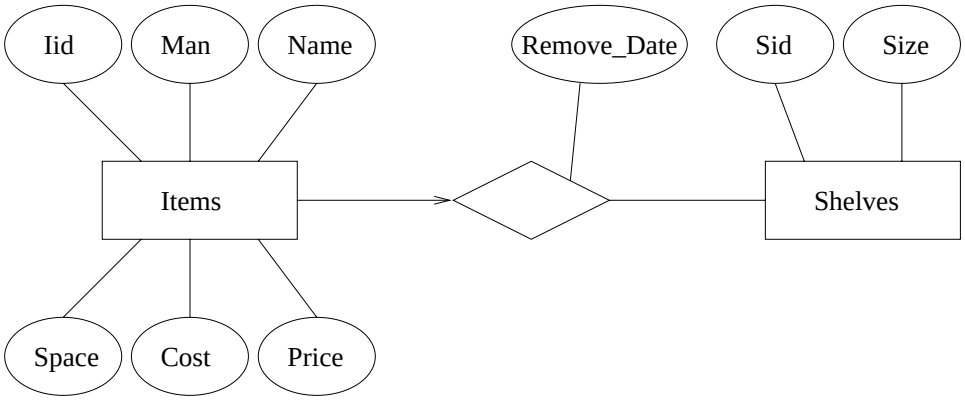
Alternatively

```
CREATE VIEW X (A3, B1) AS
        SELECT E1.A3, E2.B1
        FROM E1, R, E2
        WHERE E1.A1 = R.A1 AND E1.A2 = R.A2 AND
                        R.B1 = E2.B1
```

8. A store displays items on shelves. An item is described by a unique item Id, Iid, its manufacturer, Man, its name, Name, the space it occupies on a shelf, Space, its cost, Cost, and its retail price, Price. A shelf is described by a unique shelf Id, Sid, and its size, Size. You can assume any reasonable domains for these attributes.

   (a) Assume some items are placed on shelves. When an item is placed on a shelf a date, Remove_Date, is specified at which time the item will be removed from the shelf. Give an E-R diagram that completely describes the entities and relationships of this plan.

   Solution:



   (b) In this part you are to translate the entity and relationship sets of (8a) to tables using SQL. Show all keys (foreign and otherwise).

      i. Specify a table, SHELVES, for storing information about shelf entities.
      Solution:

```
CREATE TABLE SHELVES (
    Sid INTEGER,
    Size INTEGER,
    PRIMARY KEY Sid)
```

      ii. It is assumed that each manufacturer assigns a unique name to each item it produces and that the retail price of an item is always greater than its cost. Specify a table, ITEMS, for storing information about item entities that enforces these constraints.
      Solution:

```
CREATE TABLE ITEMS (
    Iid INTEGER,
    Man CHAR [20],
    Name CHAR [20],
    Space INTEGER,
    Cost INTEGER,
    Price INTEGER,
    PRIMARY KEY Iid,
    CHECK Price > Cost,
    UNIQUE (Man, Name))
```

      iii. Specify a table, STORES, for completely describing the relationship type.
      Solution:

```
CREATE TABLE STORES (
    Iid INTEGER,
    Sid INTEGER,
    Remove_Date DATE,
     FOREIGN KEY Sid REFERENCES SHELVES,
     FOREIGN KEY Iid REFERENCES ITEMS,
     PRIMARY KEY Iid )
```

(c) We would like the database schema to enforce the restriction that the only valid shelf sizes are 100, 150, 200, and 300. Use SQL to do this.

Solution:

Create a new schema element:

```
CREATE DOMAIN SHELF_SIZES INTEGER
        CHECK (VALUE IN (100. 150. 200, 300))
```

and change the size attribute in SHELVES to: Size SHELF_SIZES. Or add the following constraint to SHELVES

```
CHECK Size IN (100. 150. 200, 300)
```

(d) Show how the E-R diagram of (8a) changes if we add the restriction that no shelf is empty.

Solution:

The edge connecting SHELVES to the relationship becomes a heavy line.

(e) Sales people need not be concerned with the cost or item Ids of items. Create a view of Items that does not contain these attributes.

Solution:

```
CREATE VIEW Sales (Man, Name, Space, Price) AS
    SELECT Man, Name, Space, Price
    FROM ITEMS
```

Problems for Chapter 5: Relational Algebra and SQL

1. Consider the following schema:

   BOOK(ISBN,Title,Publisher,PublicationDate)
   AUTHOR(AName,Birthdate)
   PUBLISHER(Pname,Address)
   WROTE(ISBN,AName) // which author wrote which book

   Use the **relational algebra** to express the following queries:

   (a) Find all book titles published by Acme Publishers
   (b) Find all authors of the book with ISBN 0444455551
   (c) Find all authors who published at least one book with Acme Publishers
   (d) Find all authors who never published a book with Acme Publishers.

   Solution:

   (a) $\pi_{\texttt{Title}}(\sigma_{\texttt{PName='Acme'}}(\text{BOOK}))$
   (b) $\pi_{\texttt{AName}}(\sigma_{\texttt{ISBN='0444455551'}}(\text{WROTE}))$
   (c) $\pi_{\texttt{AName}}(\ (\sigma_{\texttt{PName='Acme'}}(\text{BOOK}))\ \bowtie\ \text{WROTE})$
   (d) $\pi_{\texttt{AName}}(\text{AUTHORS}) - \pi_{\texttt{AName}}(\sigma_{\texttt{PName='Acme'}}(\text{BOOK}) \bowtie\ \text{WROTE})$

2. Consider the following schema:

   MANUFACTURER(MName,HQAddress,Country)
   CAR(Make,ModelNum,MName,Kind)
   DEALERSHIP(DName,DAddress)
   SELL(DName,Make,ModelNum)

   Use the **Relational Algebra** to express the following queries:

   (a) Find the makes of all cars manufactured by "Mercedes/Chrysler"

   (b) Find the names and addresses of all dealers who sell Plymouths (i.e., cars with make 'Plymouth'.)

   (c) Find the addresses of dealers that sell at least one make of car manufactured by a company incorporated in Japan.

   (d) Find all dealers who don't sell any car manufactured by "Mercedes/Chrysler".

   Solution:

   (a) $\pi_{\texttt{Make}}(\sigma_{\texttt{MName='Mercedes/Chrysler'}}(\text{CAR}))$

   (b) $\pi_{\texttt{DName,DAddress}}(\text{DEALERSHIP} \bowtie \sigma_{\texttt{Make='Plymouth'}}(\text{SELL}))$

   (c) $\pi_{\texttt{DAddress}}(((\sigma_{\texttt{Country='Japan'}}(\text{MANUFACTURER}) \bowtie \text{CAR}) \bowtie \text{SELL}) \bowtie \text{DEALERSHIP})$

   (d) $\pi_{\texttt{DName}}(\text{DEALERSHIP}) - \pi_{\texttt{DName}}(\sigma_{\texttt{MName='Mercedes/Chrysler'}}(\text{CAR}) \bowtie \text{SELL})$

3. Consider the following SQL query:

```
SELECT  P.Name, C.Name
FROM    Professor P, Course C, Taught T
WHERE   P.Id = T.ProfId  AND  T.Semester = 'S2002'
        AND T.CrsCode = C.CrsCode
```

Write down an equivalent expression in relational algebra.

Solution:

$\pi_{\texttt{Professor.Name,Course.Name}}(\text{PROFESSOR} \bowtie_{\texttt{Id=ProfId}} \sigma_{\texttt{Semester='S2002'}}(\text{TAUGHT}) \bowtie_{\texttt{CrsCode=CrsCode}} \texttt{Course})$

4. Given the following schema:

   STUDENT(Id,Name)
   TRANSCRIPT(StudId,CourseName,Semester,Grade)

   Formulate the following query in SQL: *Create a list of all students (Id, Name) and, for each student, list the average grade for the courses taken in the* S2002 *semester.*
   Note that there can be students who did not take any courses in S2002. For these, the average grade should be listed as 0.

   Solution:

   We first create a view which augments TRANSCRIPT with rows that enroll every student into a NULL course with the grade of 0. Therefore, students who did not take anything in semester 'S2002' will have the average grade of 0 for that semester.

```
CREATE VIEW TranscriptView AS (
    ( SELECT * FROM Transcript)
    UNION
    ( SELECT S.Id,NULL,'S2002',0
      FROM Student S)
      WHERE S.Id NOT IN ( SELECT T.StudId
                          FROM Transcript T
                          WHERE T.Semester = 'S2002') )
    )

SELECT S.Id, S.Name, AVG(T.Grade)
FROM Student S, TranscriptView T
WHERE S.Id = T.StudId AND T.Semester = 'S2002'
GROUP BY S.Id
```

5. Consider the following relation schemas:

   CARS (Make,Model,Year,VINNumber)
   CUSTOMERS (Id,Name)
   RENTALS (Id,VINNumber,Date)

   Use the **relational algebra** and the **division operator** to answer the following query:

   Find all customer names who have previously rented every Toyota made in year 2000.

   Solution:

   $\pi_{\texttt{Name,VINNumber}}(\text{CUSTOMERS} \bowtie \text{RENTALS}) \;/\; \pi_{\texttt{VINNumber}}(\sigma_{\texttt{Make='Toyota' AND Year='2000'}}(\text{CARS}))$

6. Consider the following schema:

   EMPLOYEE (ID,Name,Address)
   SUPPLIER (ID,Name)
   PURCHASEORDER (OrderID,EmpIssuerID,SupplierID,Date)
   PURCHASEITEM (ItemID,OrderID,ItemName,ItemCost)

   The first two schemas are self-explanatory. Each tuple in PURCHASEORDER describes a purchase order issued by a particular employee to a particular supplier. The last relation, PURCHASEITEM, describes each ordered item and its relationship to the corresponding order.

   Write the following queries using the SQL language:

   (a) Names of employees who have made a purchase order that contains an item costing more than $150.

   Solution:

```
SELECT E.Name
FROM EMPLOYEE E, PURCHASEORDER P, PURCHASEITEM I
WHERE E.ID=P.EmpIssuerID AND P.OrderID = I.OrderID
        AND ItemCost > 150
```

   (b) For each supplier, list the name and the total cost of all items ever ordered from this supplier.

   Solution:

```
SELECT S.Name, SUM(I.ItemCost)
FROM SUPPLIER S, PURCHASEORDER P, PURCHASEITEM I
WHERE S.ID=P.SupplierID AND P.OrderID = I.OrderID
GROUP BY P.SupplierID
```

(c) Number of orders such that the total cost of items in each of those orders is over $200.

Solution:

```
SELECT  COUNT(P.OrderID)
FROM  PurchaseOrder P
WHERE 200 < ( SELECT SUM(I.ItemCost)
                FROM  PurchaseItem I
                WHERE I.OrderID = P.OrderID
                GROUP BY I.OrderID  )
```

(d) Names of employees who have issued a purchase order to *every* supplier.

Solution:

```
SELECT E.Name
FROM  Employee E
WHERE  NOT EXISTS (
            ( SELECT  S.Id
              FROM  Supplier S )
             EXCEPT
            ( SELECT P.SupplierID
              FROM  PurchaseOrder P
              WHERE E.Id = P.EmpIssuerID)
         )
```

7. Consider the following database schema, where the keys are underlined:

   FLIGHT(FltNumber,From,To,DepartureDateTime,ArrivalDateTime)
   TICKET(Id,TravelAgent,Passenger)
   ITINERARY(TicketNum,FltNumber)

   (a) Use **both** the relational algebra **and** SQL to answer the following query:

   Find all possible trips from LA to NYC, which consist of two connecting flights. (Flights connect if flight 1 arrives at the airport from where flight 2 leaves and the arrival time of flight 1 is less than the departure time of flight 2. You can use < to compare the times.)

   Solution:

   SQL:

```
SELECT  *
  FROM  FLIGHT F1,  FLIGHT F2
  WHERE F1.To = F2.From  AND F1.ArrivalDateTime < F2.ArrivalDateTime
        AND F1.From = 'LA'  AND F2.To = 'NYC'
```

   Algebra:

   $\sigma_{\texttt{From='LA'}}(\text{FLIGHT}) \bowtie_{\texttt{To=From AND ArrivalDateTime < DepartureDateTime}} \sigma_{\texttt{To='NYC'}}(\text{FLIGHT})$

   (b) Use relational **algebra only** to answer the following query:

   Find the travel agents who issued a ticket for **every** *flight originating in LA.*

   Solution:

   $\pi_{\texttt{FltNumber, TravelAgent}}\ (\text{TICKET} \bowtie_{\texttt{Id=TicketNum}}\ \text{ITINERARY})\ /\ \pi_{\texttt{FltNumber}}(\sigma_{\texttt{From='LA'}}\ (\text{FLIGHT}))$

(c) Use SQL for the following query:

Find the travel agents who sold more than 5 tickets.

Solution:

```
SELECT  T.TravelAgent
FROM  TICKET T
WHERE 5 < ( SELECT COUNT(*)
            FROM  TICKET T1
            WHERE T1.TravelAgent = T.TravelAgent)
```

An even better solution is

```
SELECT  T.TravelAgent, COUNT(*)
FROM  TICKET T
GROUP BY T.TravelAgent
HAVING  COUNT(*) > 5
```

(d) Use SQL (only) to

Find the travel agents who sold the most number of tickets (among all the agents).

Solution:

```
SELECT T.TravelAgent
  FROM  TICKET T
  WHERE ( SELECT COUNT(*)
          FROM  TICKET T1
          WHERE T1.TravelAgent = T.TravelAgent)
        =
        ( SELECT Max(*)
            FROM  TICKET T2
            GROUPBY T2.TravelAgent)
```

8. Assume the following tables, describing employees and the departments in which they work, for all the parts of this question.

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

(a) Write a SELECT statement that outputs the employee id (eid) of all employees that work in the department with department id (did) equal to 5.

Solution:

```
SELECT W.eid
FROM   WORKS W
WHERE  did = 5
```

(b) Write a SELECT statement that outputs the names (ename) and salaries (salary) of all employees that work in the department with department name (dname) "accounting".

Solution:

```
SELECT E.ename, E.salary
FROM   EMP E,  WORKS W,  DEPT D
WHERE  E.eid = W.eid  AND W.did = D.did  AND
            D.dname = ''accounting''
```

(c) Write a SELECT statement that for each department outputs the department name (dname) and the number of employees that work in that department.

Solution:

```
SELECT  D.dname,  COUNT(*)
FROM  WORK W,  DEPT D
WHERE W.did = D.did
GROUP BY D.dname, D.did
```

(d) Write the declaration of a view that gives the name of each employee who works in more than one department.

Solution:

```
CREATE VIEW multiple-depts  AS
    SELECT E.name
        FROM  EMP E,  WORKS W
        WHERE E.eid = W.eid
        GROUP BY E.eid, E.name
        HAVING COUNT (*) > 1
```

or

```
CREATE VIEW multiple-depts  AS
```

```
SELECT E.name
FROM Emp E
WHERE (SELECT COUNT (*)
    FROM Works W
    WHERE W.eid = E.eid) >1
```

9. Use the following schema for this problem:

```
SAILORS(sid: integer, sname: string, rating: integer, age: real)
       key - sid
BOATS(bid: integer, bname: string, color: string)
       key - bid
RESERVES(sid: integer, bid: integer, day: date)
       key - sid, bid, day
```

(a) Output the names of all sailors over 20.

Solution:

```
SELECT S.sname
FROM SAILORS S
WHERE S.age > 20
```

(b) Output the names of all sailors who have reserved red boats.

Solution:

```
SELECT S.sname
FROM SAILORS S
WHERE S.sid IN
      ( SELECT R.sid
        FROM RESERVES R, BOATS B
        WHERE B.bid = R.bid AND B.color = ‘‘red’’)
```

(c) Output the names of all sailors who have made exactly five reservations.

Solution:

```
SELECT S.sname
FROM SAILORS S
WHERE S.sid IN
      ( SELECT R.sid
        FROM RESERVES R
        GROUP BY R.sid
        HAVING COUNT(*) = 5 )
```

10. Use the following schema for this problem:

```
Sailors(sid: integer, sname: string, rating: integer, age: real)
        key - sid
Boats(bid: integer, bname: string, color: string)
        key - bid
Reserves(sid: integer, bid: integer, day: date)
        key - sid, bid, day
```

(a) Output the names of all sailors who have only rented a single boat (although they might have rented that boat several times).

Solution:

```
SELECT sname
FROM Sailors S
WHERE ( SELECT COUNT DISTINCT R.bid
                FROM Reserves R
                WHERE R.sid = S.sid) = 1
```

(b) Define a view giving the reservation history of junior sailors (sailors under 16 years old). For each reservation that a junior sailor has made, the view has a row giving the sailor's name and Id, the boat name and Id, and the day the boat was reserved.

Solution:

```
CREATE VIEW V AS
    SELECT S.sid, S.sname, B.bid, B.name, R.day
    FROM Sailors S, Reserves R, Boats B
    WHERE S.sid = R.sid AND R.bid = B.bid AND S.age < 16
```

(c) Using the view of (10b), give a SELECT statement that returns the name of each sailor, the name of the boat that that sailor has rented, and the number of times the sailor has rented that boat.

Solution:

```
SELECT V.sname, V.bname, COUNT (*)
FROM V
GROUP BY V.sid, V.sname, V.bname, V.bid
```

11. Given the following schema:

TEACHING(ProfId, CrsCode, Semester)
PROFESSOR(Id, Name, DeptId)
TRANSCRIPT(StudId, CrsCode, Semester, Grade)
STUDENT(Id, Name, Address, Status)

(a) Write a SELECT statement that outputs the course codes of all courses in which the student with Id 111111 is registered for the F2000 semester.

Solution:

```
SELECT  T.CrsCode
FROM  TRANSCRIPT T
WHERE  T.Semester = 'F2000'  AND T.Student = 111111
```

(b) Write a SELECT statement that outputs the names of all professors that have taught both CSE305 and CSE315.

Solution:

```
SELECT  P.Name
FROM   PROFESSOR P,  TEACHING T,  TEACHING T1
WHERE P.Id = T.ProfId  AND P.Id = T1.ProfId
    AND T.CrsCode = 'CSE305'  AND T1.CrsCode = 'CSE315'
```

(c) Write a SELECT statement that lists, for each professor that taught CSE305, the professor's name and the number of times he/she has taught the course.

Solution:

```
SELECT P.Name,  COUNT (*)
FROM  PROFESSOR P,  TEACHING T
WHERE P.Id = T.ProfdId  AND T.CrsCode = 'CSE305'
GROUP BY P.Name, P.Id
```

(d) In order to help a student with courses he/she will be taking in F2000 we want to output a set of professors who can provide tutoring. Write a SELECT statement that, for each student, outputs the set of professor Ids of professors that have taught a course in which the student is registered in F2000. (Hint: use a correlated nested subquery based on your answer to (11a)).

Solution:

```
SELECT S.StudId, R.ProfId
FROM  TEACHING R,  STUDENT S
WHERE R.CrsCode  IN
       ( SELECT T.CrsCode
         FROM  TRANSCRIPT T
         WHERE T.Semester = 'F2000'  AND T.StudId = S.StudId)
```

12. Assume a schema for the following set of relations (from problem (8b) of the exam problems for Section):

```
SHELVES(Sid, Size) with primary key Sid
ITEMS(Iid, Man, Name, Space, Cost, Price) with primary key Iid
STORES(Iid, Sid, Remove_Date) with primary key Iid
```

(a) Give a SELECT statement that returns all rows of SHELVES.
Solution:

```
SELECT *
FROM SHELVES
```

(b) Give a SELECT statement that returns all rows of SHELVES corresponding to shelves that have a size greater than 100.
Solution:

```
SELECT *
FROM SHELVES
WHERE Size > 100
```

(c) Give a SELECT statement that returns the manufacturer and name in all rows of ITEMS corresponding to items whose retail price is more than twice its cost.
Solution:

```
SELECT Man, Name
FROM ITEMS
WHERE Price > 2 * Cost
```

(d) Give a SELECT statement that returns the manufacturer and name in all rows of ITEMS corresponding to items stored on the shelf whose Sid has value 12345. **Solution:**

```
SELECT I.Man, I.Name
FROM ITEMS I, STORES S, SHELVES V
WHERE I.Iid = S.Iid AND S.Sid = V.Sid AND S.Sid = 12345
```

(e) We would like the database schema to enforce the restriction that, for any item stored on a shelf, the space occupied by the item not exceed the size of the shelf. Use SQL to do this.
Solution:

```
CREATE ASSERTION FITS
    CHECK ( NOT EXISTS (
        SELECT *
        FROM ITEMS I, STORES S, SHELVES V
        WHERE I.Iid = S.Iid AND S.Sid = V.Sid
              AND I.Space > V.Size))
```

13. Consider the following schema:

PROFESSOR(Id: INTEGER,Name: STRING,DeptId:DEPTS)
TEACHING(ProfId:INTEGER,CrsCode:COURSES,Semester:SEMESTERS)
TRANSCRIPT(StudId:INTEGER,CrsCode:COURSES,Semester:SEMESTERS,Grade:GRADES)

(a) Write a SELECT statement that returns the course code of every course that has ever been taught by a professor whose DeptId is 'CS'.

Solution:

```
SELECT DISTINCT T.CrsCode
FROM TRANSCRIPT T, PROFESSOR P
WHERE P.DeptId = 'CS' AND P.Id = T.ProfId
```

(b) Translate the above statement into an equivalent relational algebra expression. Briefly describe the evaluation procedure that your expression specifies.

Solution:

$$\pi_{CrsCode}(\sigma_{DeptId='CS'}(Teaching \bowtie_{ProfId=Id} Professor))$$

or

$$\pi_{CrsCode}(\sigma_{DeptId='CS'}(Professor) \bowtie_{ProfId=Id} Teaching)$$

others are possible.

(c) Translate the following relational algebra expression into an SQL statement.

$$\pi_{Grade}(\sigma_{Semester='F2001\ AND\ ProfId=9999}(Transcript \bowtie Teaching))$$

Solution:

```
SELECT R.StudId
FROM TRANSCRIPT R, TEACHING E
WHERE E.ProfId = 9999 AND R.CrsCode = E.CrsCode
    AND R.Semester = E.Semester
```

14. Consider the following schema:

TAKES(Student, Course)
OFFERS(Department, Course)

Write the following query in SQL: *Find all students who took all their courses only in the CS department*.

Solution:

```
SELECT T.Student
FROM   TAKES T
WHERE  NOT EXISTS (
                    ( SELECT TT.Course
                      FROM   TAKES TT
                      WHERE TT.Student = T.Student)
                    EXCEPT
                    ( SELECT O.Course
                      FROM   OFFERS O
                      WHERE O.Department = 'CS')
                  )
```

15. Consider the following schema:

    STUDENT(Id: STRING,Name:STRING,Address:STRING,Status:STRING)
    TRANSCRIPT(StudId:STRING,CrsCode:COURSES,Semester:SEMESTERS,Grade:GRADES)

    (a) Write an SQL statement that returns the set of course codes of all CS courses that have ever been taught. Assume that the course code of a CS course begins with 'CS'.

    Solution:

```
SELECT T.CrsCode
FROM  TRANSCRIPT T
WHERE T.CrsCode  LIKE 'CS%'
```

    (b) Write an SQL statement that returns the set of course codes of all CS courses taken by the student with StudId '111111111'.

    Solution:

```
SELECT T.CrsCode
FROM  TRANSCRIPT T
WHERE T.CrsCode  LIKE 'CS%'  AND T.StudId = '111111111'
```

    (c) Use the above results (perhaps with some small modification) to write an SQL statement that gives the names of all students who have taken all CS courses.

    Solution:

```
SELECT T.CrsCode
FROM  STUDENT S
WHERE NOT EXISTS
    ( SELECT T1.CrsCode
      FROM  TRANSCRIPT T1
      WHERE T1.CrsCode  LIKE 'CS%
          EXCEPT
      SELECT T2.CrsCode
      FROM  TRANSCRIPT T2
      WHERE T2.CrsCode  LIKE 'CS%'  AND S.Id = T2.StudId)
```

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

1. Consider a schema with the attribute set ABCDFG and the following FDs: AB → CD, BC → FG, A → G, G → B, C → G.

   (a) Find a minimal cover of this set of FDs.

   (b) Is the decomposition of the previous schema into ABCD and CFG lossless?

   Solution:

   (a) **Step 1**

   AB → C
   AB → D
   BC → F
   BC → G
   A → G
   G → B
   C → G

   Step 2 - Reduce the left-hand sides

   Since A^+ = AGBCDF (and so A→B is entailed), we can replace the first two FDs with A → C, A → D.

   Similarly, C^+ = CGBFG, so FDs 3 and 4 can be replaced with C → F, C → G (the latter is a duplicate and can be deleted).

   Step 3 - eliminate redundant FDs

   A → G is redundant due to A → C, C → G. So, we end up with the following minimal cover:

   A → C
   A → D
   C → F
   G → B
   C → G

   (b) The decomposition into ABCD and CFG is lossless, since the intersection is C and C^+ = CFGB. In particular, the FDs imply C → CFG, which implies losslessness according to the losslessness criteria.

2. Consider the following database schema. The attributes are ABCDEGHKLM (10 in total). The FDs are:

    1. ABE → CK
    2. AB → D
    3. C → BE
    4. EG → DHK
    5. D → L
    6. DL → EK
    7. KL → DM

    (a) Compute the attribute closure of EGL with respect to the above set of FDs. Show all steps.

    Solution:

    ```
    EGL
    EGLDHK              by 4
    EGLDHKL             by 5
    EGLDHKLM            by 7
    ```

(b) Compute the minimal cover of the above set of FDs. Show all steps.

Solution:

Step 1: Reduce the right-hand sides

1. ABE → C
2. ABE → K
3. AB → D
4. C → B
5. C → E
6. EG → D
7. EG → H
8. EG → K
9. D → L
10. DL → E
11. DL → K
12. KL → D
13. KL → M

Step 2: Reduce the left-hand sides

Since AB^+ includes E, E can be eliminated from 1 and 2.

Similarly, L can be eliminated from 10 and 11. Thus, we obtain

1. AB → C
2. AB → K
3. AB → D
4. C → B
5. C → E
6. EG → D
7. EG → H
8. EG → K
9. D → L
10. D → E
11. D → K
12. KL → D
13. KL → M

Step 3: Eliminate redundant FDs

FD 2 follows from 3 and 11. Nothing else can be eliminated, so the minimal cover is

FD8 follows from 6 and 11.

1. AB → C
3. AB → D
4. C → B
5. C → E
6. EG → D
7. EG → H
9. D → L

10. D → E
11. D → K
12. KL → D
13. KL → M

(c) Use the 3NF synthesis algorithm to construct a *lossless, dependency preserving* decomposition of the above schema. Show all steps.

Solution:

$\mathbf{R}_1$ = (ABCD, {AB → CD})
$\mathbf{R}_2$ = (CBE, {C → BE})
$\mathbf{R}_3$ = (EGDH, {EG → DH})
$\mathbf{R}_4$ = (DLEK, {D → LEK})
$\mathbf{R}_5$ = (KLDM, {KL → DM})

(d) Are all the schemas in the resulting decomposition in BCNF? If yes, then you are done. If there are schemas that are not in BCNF, decompose them further to achieve BCNF. Is the resulting decomposition dependency-preserving? Yes/no answers do not count. Explain everything.

Solution:

$\mathbf{R}_1$ is not in BCNF due to C → B: Split into CB and CDA.

$\mathbf{R}_3$ is not in BCNF due to D → E: Split into DE and DHG.

Note: $\mathbf{R}_5$ **is** in BCNF spite D → KL, because D is a key of $\mathbf{R}_5$ along with KL.

The resulting decomposition is *not* dependency-preserving: The split of $\mathbf{R}_1$ looses AB → CD and the split of $\mathbf{R}_3$ looses EG → DH.

3. Consider the schema **R** over the attributes KNLMDFG with the following functional dependencies:

$$\begin{aligned}
&\texttt{KN} \rightarrow \texttt{L}\\
&\texttt{LM} \rightarrow \texttt{N}\\
&\texttt{LMN} \rightarrow \texttt{D}\\
&\texttt{D} \rightarrow \texttt{M}\\
&\texttt{M} \rightarrow \texttt{FG}
\end{aligned}$$

and the following multivalued dependencies:

$$\begin{aligned}
\mathbf{R} &= \texttt{NLK} \bowtie \texttt{NDMFG}\\
\mathbf{R} &= \texttt{KNMF} \bowtie \texttt{FLDG}
\end{aligned}$$

Decompose this schema into 4NF using the following method: first obtain a BCNF decomposition using the FDs only. Then proceed to apply the MVDs to further normalize the schemas that are not yet in 4NF.

Solution:

The minimal cover is

$$\begin{aligned}
&\texttt{KN} \rightarrow \texttt{L}\\
&\texttt{LM} \rightarrow \texttt{N}\\
&\texttt{LM} \rightarrow \texttt{D}\\
&\texttt{D} \rightarrow \texttt{M}\\
&\texttt{M} \rightarrow \texttt{F}\\
&\texttt{M} \rightarrow \texttt{G}
\end{aligned}$$

So 3NF is

$$\begin{aligned}
\mathbf{R}_1 &= (\texttt{KNL},\ \texttt{KN} \rightarrow \texttt{L})\\
\mathbf{R}_2 &= (\texttt{LMND},\ \texttt{LM} \rightarrow \texttt{ND})\\
\mathbf{R}_3 &= (\texttt{DM},\ \texttt{D} \rightarrow \texttt{M})\\
\mathbf{R}_4 &= (\texttt{MFG},\ \texttt{M} \rightarrow \texttt{FG})
\end{aligned}$$

This is not yet lossless, since none of the attribute sets of $\mathbf{R}_i$ is a superkey of **R**. So, we need to add a new schema whose attribute set is a superkey: $\mathbf{R}_0$ = (KNLD, { }). (Actually, this schema has some implied FDs, such as KN→L.)

$\mathbf{R}_2$ is not in BCNF due to D → M. Split into $\mathbf{R}_{21}$ = DM (same as $\mathbf{R}_3$) and $\mathbf{R}_{22}$ =DLN.

We now need to check which of the resulting schemas are still not in 4NF and decompose them with respect to the MVDs.

$\mathbf{R}_1$ and $\mathbf{R}_{21}$ are in 4NF.

$\mathbf{R}_{22}$ is *not* in 4NF due to MVD 1. Splitting into NL and ND (with no FDs embedded into these schemas).

$\mathbf{R}_3$ is in 4NF.

$\mathbf{R}_4$ is *not* in 4NF due to MVD 2. Splitting into MF and FG.

The resulting 4NF decomposition has schemas with these attribute sets: KNL, DM, NL, ND, MF, and FG.

4. Consider the following database schema. The attributes are ABCDEF. The FDs are:

   ABF → C
   CF → B
   CD → A
   BD → AE
   C → F
   B → F

   (a) Compute the attribute closure of ABD with respect to the above set of FDs. Show all steps.

   Solution:

   Using the attribute closure algorithm:

ABD		
ABDE	by	BD->E
ABDEF	by	B->F
ABDEFC	by	ABF->C

(b) Compute the minimal cover of the above set of FDs. Show all steps.

Solution:

Step 1: Replace $\texttt{BD} \rightarrow \texttt{AE}$ with $\texttt{BD} \rightarrow \texttt{A}$ and $\texttt{BD} \rightarrow \texttt{E}$.

Step 2: $\texttt{ABF} \rightarrow \texttt{C}$ can be replaced with: $\texttt{AB} \rightarrow \texttt{C}$ while $\texttt{CF} \rightarrow \texttt{B}$ can be replaced with: $\texttt{C} \rightarrow \texttt{B}$. This is because (1) $\texttt{AB} \rightarrow \texttt{C}$ is entailed by $\texttt{ABF} \rightarrow \texttt{C}$ and $\texttt{B} \rightarrow \texttt{F}$; and $\texttt{C} \rightarrow \texttt{B}$ is entailed by $\texttt{CF} \rightarrow \texttt{B}$ and $\texttt{C} \rightarrow \texttt{F}$; (2) $\texttt{AB} \rightarrow \texttt{C}$ entails $\texttt{ABF} \rightarrow \texttt{C}$ and $\texttt{C} \rightarrow \texttt{B}$ entails $\texttt{CF} \rightarrow \texttt{B}$; and therefore (3) the set $\{\texttt{AB} \rightarrow \texttt{C},\ \texttt{B} \rightarrow \texttt{F}\}$ is equivalent to $\{\texttt{ABF} \rightarrow \texttt{C},\ \texttt{B} \rightarrow \texttt{F}\}$; and $\{\texttt{C} \rightarrow \texttt{B},\ \texttt{C} \rightarrow \texttt{F}\}$ is equivalent to $\{\texttt{CF} \rightarrow \texttt{B},\ \texttt{C} \rightarrow \texttt{F}\}$.

So, after steps 1 and 2 in the minimal cover algorithm, we have the following FDs:

$\texttt{AB} \rightarrow \texttt{C}$
$\texttt{C} \rightarrow \texttt{B}$
$\texttt{CD} \rightarrow \texttt{A}$
$\texttt{BD} \rightarrow \texttt{A}$
$\texttt{BD} \rightarrow \texttt{E}$
$\texttt{C} \rightarrow \texttt{F}$
$\texttt{B} \rightarrow \texttt{F}$

Step 3: Eliminate redundant FDs:
$\texttt{C} \rightarrow \texttt{F}$ is entailed by $\texttt{C} \rightarrow \texttt{B}$ and $\texttt{B} \rightarrow \texttt{F}$, while
$\texttt{CD} \rightarrow \texttt{A}$ is entailed by $\texttt{C} \rightarrow \texttt{B}$ and $\texttt{BD} \rightarrow \texttt{A}$. Therefore, the minimal cover is

$\texttt{AB} \rightarrow \texttt{C}$
$\texttt{C} \rightarrow \texttt{B}$
$\texttt{BD} \rightarrow \texttt{A}$
$\texttt{BD} \rightarrow \texttt{E}$
$\texttt{B} \rightarrow \texttt{F}$

(c) Use the 3NF synthesis algorithm to construct a *lossless, dependency preserving* decomposition of the above schema. Show all steps.

Solution:

The 3NF decomposition that corresponds to this minimal cover is

$\mathbf{R}_1 = (\mathtt{ABC}, \{\mathtt{AB} \rightarrow \mathtt{C}\})$
$\mathbf{R}_2 = (\mathtt{CB}, \{\mathtt{C} \rightarrow \mathtt{B}\})$
$\mathbf{R}_3 = (\mathtt{BDAE}, \{\mathtt{BD} \rightarrow \mathtt{A}, \mathtt{BD} \rightarrow \mathtt{E}\})$
$\mathbf{R}_4 = (\mathtt{BF}, \{\mathtt{B} \rightarrow \mathtt{F}\})$

Test for losslessness:
BDAE is a superkey, since $\mathtt{BDAE}^+ = \mathtt{BDAEFC}$. Therefore, the above decomposition is lossless.

(d) Are all the schemas in the resulting decomposition in BCNF? If yes, then you are done. If there are schemas that are not in BCNF, decompose them further to achieve BCNF. Is the resulting decomposition dependency-preserving? Explain all steps: yes/no answers do not count.

Solution:

The schema $\mathbf{R}_1 = (\mathtt{ABC}, \{\mathtt{AB} \rightarrow \mathtt{C}\})$ is not in BCNF because $\mathtt{C} \rightarrow \mathtt{B}$ is entailed by the original set of FDs. Since $\mathtt{C}^+ = \mathtt{CBF}$, C is not a superkey of $\mathbf{R}_1$, it follows that $\mathtt{C} \rightarrow \mathtt{B}$ violates the BCNF condition for $\mathbf{R}_1$. The BCNF decomposition (which uses $\mathtt{C} \rightarrow \mathtt{B}$) is

$\mathbf{R}_{11} = (\mathtt{CB}, \{\mathtt{C} \rightarrow \mathtt{B}\})$
$\mathbf{R}_{12} = (\mathtt{CA}, \{\})$

This decomposition is not dependency preserving, since the dependency $\mathtt{AB} \rightarrow \mathtt{C}$ no longer has a home schema and it does not follow from the dependencies associated with $\mathbf{R}_{11}$, $\mathbf{R}_{12}$, $\mathbf{R}_2$, $\mathbf{R}_3$, $\mathbf{R}_4$.

(a) What is a functional dependency?

(b) Explain why a primary key constraint is an example of a functional dependency.

Solution

The value of the primary key determines the values of all other attributes.

(c) State the conditions (in terms of functional dependencies) for a relation to be in 3^{rd} normal form and in BCNF.

(d) Consider a relation in a library system with the following attributes: author (A), title (T), copy number (C), author's year of birth (B), shelf (Sh), subject (Sub). If the library has multiple copies of a particular book they are numbered consecutively with (unique) copy numbers. Shelf is the location of a particular book and all the books on a shelf are in the same subject category. What are the functional dependencies? Decompose the relation into two or more relations to eliminate redundant storage of information. In what normal form are the relations that result from your decomposition?

Solution:

The functional dependencies are:

A T C → B Sh Sub
A → B
Sh → Sub

The relation can be decomposed into the following three relations:

(A T C Sh) with FD ATC → Sh
(A B) with FD A → B
(Sh Sub) with FD Sh → Sub

5. (a) State the conditions (in terms of functional dependencies) for a relation to be in third normal form and in BCNF.

   Solution:

   For all FD's X → A that are not trivial:

```
BCNF - (1) X is a superkey

3rd NF - (1) X is a superkey
         (2) A is contained in a key
```

   (b) Consider a relation having attributes ABCDE in which the only non-trivial FDs are D → C and AB → CDE. What is the key of the relation? Is it in 3rd NF? Explain.

   Solution:

   AB is the key. The relation is not in 3rd NF since D → C violates (1) and (2)

   (c) Decompose the relation of (5b) into smaller relations which are in BCNF. Give the keys of the resulting relations.

   Solution:

   R1: ABDE with key AB
   R2: DC with key D

   (d) What is the necessary and sufficient condition for a decomposition to be lossless? Is your decomposition in (5c) lossless?

   Solution:

   A decomposition of **R** into **R1** and **R2** is lossless if the common attributes of **R1** and **R2** are a key of either **R1** or **R2**. The decomposition of (5c) is lossless.

6. Assume a relation schema, R, that has attributes A, B, C, D, and E. The only functional dependencies (FD's) are AB → CDE, D → E, and CD → B.

   (a) What is a key of R?

   Solution:

   AB

   (b) What does the FD D → E mean?

   Solution:

   For each value of D there can be at most one value of D.

   (c) What does the FD D → E imply about the redundant storage of information in an instance of R? Explain.

   Solution:

   If there are a number of rows in an instance of R with the same value of D they will all have the same value of E. This means that the value of E is stored multiple times unnecessarily.

   (d) The insertion of a new row in R might cause a violation of an FD. In order to prevent this for the FD D → E we might try one of the following.

      i. We might add the constraint UNIQUE (D,E). Would this be an appropriate was to enforce the FD? Explain your answer.
      Solution:
      No. Since D is not a key there can be multiple rows with the same value of D and hence the same value of E. The constraint would prevent this although multiple rows with the same value of DE models the real world.

      ii. Write an assertion that prevents a violation of the FD from taking place.
      Solution

```
CREATE ASSERTION X
    CHECK ((SELECT MAX COUNT DISTINCT E
                FROM R
                GROUP BY D ) <= 1)
```

   or

```
CREATE ASSERTION X
    CHECK NOT EXISTS
                ( SELECT *
                  FROM R R1, R R2
                  WHERE R1.D = R2.D AND R1.E <> R2.E)
```

   (e) Is R in 3rd Normal Form? Explain.

   Solution:

   No. The FD D → E violates the conditions for 3rd Normal Form.

      i. {*E*} is not contained in {*D*}
      ii. D is not a superkey
      iii. E is not part of a key

(f) What is the condition for losslessness? Explain why it is important for a decomposition to be lossless?

Solution:

A decomposition of R into R1 and R2 is lossless if $R = R1 \bowtie R2$ and either $R1 \bigcap R2 \rightarrow R1$ or $R1 \bigcap R2 \rightarrow R2$

(g) Give a lossless decomposition of R into two 3^{rd} Normal Form relations R1 and R2. Demonstrate that R1 and R2 are in 3^{rd} Normal Form by showing that the FD's that apply to each satisfy the conditions for 3^{rd} NF. Show that the decomposition is lossless.

Solution:

R1: ABCD with FDs $\texttt{AB} \rightarrow \texttt{CD}, \texttt{CD} \rightarrow \texttt{B}$
R2: DE with FD $\texttt{D} \rightarrow \texttt{E}$
R1 is in 3^{rd} Normal Form since AB is a key and B is part of a key.
R2 is in 3^{rd} Normal Form since D is a key.
lossless: $\{ABCD\} \bigcap \{DE\} = \{D\}$ and D is a key of R2.

(h) Is R1 in BCNF? Is R2 in BCNF? Explain.

Solution:

R1 is not in BCNF because of FD $\texttt{CD} \rightarrow \texttt{B}$ since CD is not a superkey.
R2 is in BCNF since D is a key.

7. (a) If the functional dependency $\mathtt{X} \rightarrow \mathtt{Y}$ is a key constraint, what are X and Y?

   Solution:

   X is the key and Y is the set of all attributes.

   (b) A trivial functional dependency is one which satisfies $\mathtt{X} \subseteq \mathtt{Y}$. Can a key be the set of all attributes if there is at least one non-trivial functional dependency in a schema? Explain.

   Solution:

   Suppose a key, k, contained all attributes. Suppose there is a non-trivial FD $\mathtt{X} \rightarrow \mathtt{Y}$. Since k contains all attributes, it contains X and Y. But the FD says that X determines Y. Hence we can delete $\mathtt{Y} - (\mathtt{X} \bigcap \mathtt{Y})$ from k and the remaining attributes still form a key. This means that k is not minimal, a contradiction.

8. Use the schema R in the following questions:

$$R = (ABCDEFGH, \{BE \rightarrow GH, G \rightarrow FA, D \rightarrow C, F \rightarrow B\})$$

(a) What is a key of R?

Solution:

BED or DEG or FED

(b) What is the attribute closure of GH ?

Solution:

$(GH)^+ = GHFAB$

(c) Can there be a key that does not contain D? If such a key exists, give it; otherwise explain why it is not possible.

Solution: No - since D is not uniquely determined by any other set of attributes, it must be a part of the key (since the key uniquely determines all attributes).

(d) Is the schema in BCNF? Give the reason for your answer.

Solution:

No - none of the LHSs are superkeys.

(e) Use one cycle of the BCNF synthesis algorithm to decompose R into two sub-relations. For each sub-relation, explain whether or not it is in BCNF.

Solution:

Four possible decompositions are.

i. $R1 = (ABDEFGH, \{BE \rightarrow GH, G \rightarrow FA, F \rightarrow B\})$
$R2 = (DC, \{D \rightarrow C\})$
R2 is in BCNF since D is a key. R1 is not in BCNF since the LHS of all FDs are not keys.

ii. $R3 = (BCDEGH, \{BE \rightarrow GH, D \rightarrow C\})$
$R4 = (GFA, \{G \rightarrow FA\})$
R4 is in BCNF since G is a key. R3 is not in BCNF since neither BE or D is a key.

iii. $R5 = (ACDEFGH, \{G \rightarrow FA, D \rightarrow C\})$
$R6 = (FB, \{F \rightarrow B\})$
R6 is in BCNF since F is a key. R5 is not in BCNF since neither D or G is a key.

iv. $R7 = (ABCDEF, \{F \rightarrow B, D \rightarrow C\})$
$R8 = (BEGH, \{BE \rightarrow GH\})$
R8 is in BCNF since BE is a key. R7 is not in BCNF since neither F nor D is a key.

(f) Give the general condition that guarantees that a decomposition is lossless. Apply the condition to your decomposition to show whether or not it is lossless.

Solution:

(intersection of attribute sets of two components) $\rightarrow$ (key of one component)
In all four decompositions the intersection of the attribute sets is the key of one of the components.

(g) Give the general condition that guarantees that a decomposition is dependency preserving. Apply the condition to your decomposition to show whether or not it is dependency preserving.

Solution:

If L is the set of FDs in R and L1 and L2 are the sets of dependencies in the components then $\mathtt{L}^+ = (\mathtt{L1} \bigcup \mathtt{L2})^+$ In the first decomposition $L = L1 \bigcup L2$ so the condition holds. The condition is not true for the other decompositions. For example, in the last decomposition we see that the attribute closure of G using $\{\mathtt{F} \rightarrow \mathtt{B}, \mathtt{D} \rightarrow \mathtt{C}, \mathtt{BE} \rightarrow \mathtt{GH}\}$ does not contain FA (it is just G).

(h) * Consider a schema $(\overline{R}, \mathcal{F})$, where $\overline{R}$ is a set of attributes and $\mathcal{F}$ is a set of functional dependencies. Assume further that

- every dependency in $\mathcal{F}$ has the form $\overline{X} \rightarrow \overline{Y}$, where the left-hand side, $\overline{X}$, consists of a *single attribute*. For instance, $B \rightarrow CD$, where B, C, D are attributes, would be an acceptable dependency.
- There is an attribute, $A_0 \in \overline{R}$, which does not belong to *any* key of the schema.

i. Prove that if this schema is in the Third Normal Form (3NF) then it is also in Boyce-Codd Normal Form (BCNF).

ii. Find a relation that satisfies all the assumptions above *except* the assumption about the existence of A_0, which is in 3NF, but not in BCNF.

Solution:

(a) Suppose, to the contrary, that the schema is not in BCNF. Then there must be an FD of the form $\overline{X} \rightarrow C \in \mathcal{F}^+$ such that $\overline{X}$ is not a superkey, $C \in \overline{K}$, for some key $\overline{K}$.

Since $\overline{K}$ is a key, there must be an FD of the form $\overline{Y} \rightarrow A_0 \in \mathcal{F}$, where $\overline{Y} \subseteq \overline{K}$ (where A_0 is the attribute that does not belong to any key). This follows from the algorithm for computing the closure of an attribute set.

But since the left-hand sides of the FDs in $\mathcal{F}$ are singleton attributes, it follows that $\overline{Y}$ is a singleton attribute. Furthermore, $\overline{Y}$ must be equal $\overline{K}$ or else $\overline{Y} \rightarrow A_0$ violates 3NF (since A_0 does not belong to any key). Therefore, $\overline{K}$ is a set that contains only one attribute. Since $C \in \overline{K}$, it follows that $\overline{K} = \{C\}$, *i.e.*, $\{C\}$ is a key. But then, since $\overline{X} \rightarrow C \in \mathcal{F}^+$, it follows that $\overline{X}$ is a superkey – contrary to the assumption.

(b) The relation over the attributes *ABCD* with the FDs $A \rightarrow B$, $B \rightarrow A$, $C \rightarrow D$, $D \rightarrow C$. Here the keys are *AC* and *BD*. Each of the above FDs violates BCNF, but satisfies 3NF.

Problems for Chapter 7: Triggers and Active Databases

1. Consider a relation schema MyAccount(StockSymbol, Quantity, Price), which records the user's stock holdings: the name of the stock, quantity, and current price. StockSymbol is a key. The user wants a small window on her desktop to show the following view:

```
CREATE VIEW MyTotals(StockSymbol, Total) AS
SELECT M.StockSymbol, M.Quantity * M.Price
FROM MyAccount M
```

The stock broker allows the user to compute the value of the view only once a day, so periodic recomputation of the view is not an option. However, the broker allows triggers that fire whenever an update to the rows of MyAccount occurs. Write a *row-level* trigger that can provide the user with an up-to-date view of the above kind.

Solution:

```
CREATE TRIGGER StockViewTrigger
AFTER UPDATE OF Quantity, Price ON MyAccount
REFERENCING NEW AS N
FOR EACH ROW
WHEN (N.StockSymbol IN ( SELECT * FROM MyAccount))
  UPDATE MyTotals
  SET Total = N.Price*N.Quality
  WHERE StockSymbol = N.StockSymbol
```

2. Consider the following schema:

   STUDENT(Name:STRING, Id:STRING, Status:STRING)
   TRANSCRIPT(StudId:STRING,Course:STRING,Grade:FLOAT)

   A student has status "good" if and only if the average grade for all courses taken by that student is above 3.0. Write the triggers that maintain the student status field in accordance with these requirements. Namely, if the student has status "good" and the average grade falls to 3.0 or less then the status is changed to "bad." If the current status is "bad" and the average grade raises to above 3.0 then the status is changed to "good." The status should not be changed in any other case.

   Solution:

```
CREATE TRIGGER  GoodToBad
AFTER INSERT, UPDATE  ON  Transcript
REFERENCING NEW AS N
FOR EACH ROW
WHEN ( EXISTS ( SELECT S.Id
                FROM  Student S,  Transcript T
                WHERE S.Status = 'good'  AND T.StudId = N.Id
                          AND S.Id = N.Id
                GROUP BY S.Id
                HAVING avg(T.Grade) =< 3 ) )
  UPDATE  Student
  SET Status = 'bad'
  WHERE Id = N.StudId


CREATE TRIGGER  BadToGood
AFTER INSERT, UPDATE  ON  Transcript
REFERENCING NEW AS N
FOR EACH ROW
WHEN ( EXISTS ( SELECT S.Id
                FROM  Student S,  Transcript T
                WHERE S.Status = 'bad'  AND T.StudId = N.Id
                          AND S.Id = N.Id
                GROUP BY S.Id
                HAVING avg(T.Grade) > 3 ) )
  UPDATE  Student
  SET Status = 'good'
  WHERE Id = N.StudId
```

For deletion, the triggers are similar, except that we need to use REFERENCING OLD AS. For instance,

```
CREATE TRIGGER   GoodToBadWhenDeleted
AFTER DELETE  ON   Transcript
REFERENCING OLD AS  0
FOR EACH ROW
WHEN  ( EXISTS  ( SELECT  S.Id
                  FROM  Student  S,  Transcript  T
                  WHERE  S.Status = 'good'  AND  T.StudId = 0.Id
                           AND  S.Id = 0.Id
                  GROUP BY  S.Id
                  HAVING  avg(T.Grade) =< 3 ) )
  UPDATE  Student
  SET  Status = 'bad'
  WHERE  Id = 0.StudId
```

3. Consider the following schema:

   OFFERINGS(Course, Semester)
   ENROLLMENT(StudId,Course,Semester)

   Write a **statement level** trigger, which removes courses from OFFERINGS for all those courses whose enrollment falls below 10 students.

   Solution:

```
CREATE TRIGGER  WatchEnrollment
AFTER DELETE, INSERT, UPDATE  ON  Transcript
FOR EACH STATEMENT
    DELETE FROM  Offerings
    WHERE  EXISTS
              ( SELECT E.Course, E.Semester
                FROM  Enrollment E
                WHERE E.Course = Course  AND E.Semester = Semester
                GROUP BY E.Course, E.Semester
                HAVING count(E.StudId) < 10 )
```

Problems for Chapter 8: Using SQL in an Application

1. Our goal is to retrieve the name of all professors in a particular department from the Professor table given by

```
Professor(Id: INTEGER,  Name: STRING, DeptId: DEPTS)
```

The department will be supplied at run time. Assume a connection object, *con*, has been created.

(a) Write the JDBC statement used to create a prepared statement object *pre* for this query. If the statement involves a host variable, give its value at the time the JDBC statement is executed.

Solution:

```
PreparedStatement pre = con.prepareStatement
        ( ‘‘SELECT P.Name FROM Professor P WHERE P.DeptId = ?’’)
```

(b) Suppose we want to get the names of all professors in the CS Department using the prepared statement object *pre*. What JDBC statements must be executed to put these names in a results set, *res*.

Solution:

```
pre.setString(1, ‘‘CS’’) (or pre.setString(‘‘DeptId’’, ‘‘CS’’) )
ResultSet res = pre.executeQuery( )
```

(c) What JDBC statement must be executed to fetch the first name from *res*?

Solution:

```
name = res.getString(1) (or name = res.getString(‘‘Name’’) )
```

2. The parts of this problem relate to the statement:

```
SELECT x
FROM T
WHERE y = 0
```

(a) The statement is used as the basis of the declaration of a cursor named Z. Assuming the cursor is declared INSENSITIVE, what happens when it is opened?

Solution:

result set is calculated; cursor positioned to (one position before) the first element

(b) Give a FETCH statement that causes the value of x in a row of the result set to be placed in host variable w.

Solution:

```
FETCH x INTO :w
```

(c) a KEYSET DRIVEN cursor. What happens when a KEYSET DRIVEN cursor is opened? Describe a situation in which such a cursor behaves differently then the cursor of (a).

Solution:

A table containing pointers (keys) to the rows of T satisfying the cursor's WHERE clause is created.

A change to a row of T satisfying the cursor's WHERE clause that is made subsequent to opening the cursor will be seen when the row is fetched. This is not true with an INSENSITIVE cursor

(d) Suppose y is a key and we wish to execute the statement using static SQL without declaring a cursor. As before, the value of x is to be placed in host variable w. Give the static SQL statement.

Solution:

```
EXEC SQL SELECT    x
         INTO :w
         FROM T
         WHERE y=0
```

(e) Suppose the SELECT statement is constructed at run time and stored in host variable v (of type string). As before, y is a key, the value of x is to be placed in w, and no cursor is to be used. Give the dynamic SQL statements that cause this to happen.

Solution:

```
EXEC SQL PREPARE S FROM :v
EXEC SQL EXECUTE S INTO :w
```

(f) The reference to w is handled differently in static and dynamic SQL (parts (2d) and (2e)). Explain why.

Solution:

With static SQL the SQL statement is available at compile time. Hence the code for accessing w can be produced by the compiler.

With dynamic SQL the SQL statement is constructed at run time. If w were embedded in the string v, w would not be known at compile time and hence the compiler could not construct code to access it. Placing w in the EXECUTE statement makes it available to the compiler.

3. Explain the following:

   (a) Why embedded SQL requires a precompiler

   Solution:

   Because the compiler for the host language would not understand the syntax of the SQL statements

   (b) Why a database operation has to be prepared before it can be executed.

   Solution:

   The system must make an execution plan as to how to execute the statement: what indexes to use, in what order to execute the different parts of the statement, etc.

   (c) Why embedded SQL can use host variables in SQL statements and JDBC cannot.

   Solution:

   The precompiler for embedded SQL has a symbol table containing the information about the host variables so that it can generate the proper code to access and update those variables.

   (d) The difference between INSENSITIVE and KEYSET DRIVEN cursors

   Solution:

   With an INSENSITIVE cursor, when the cursor is opened, the result set is computed and stored separately from the table. It can then be accessed with FETCH statements.

   With a KEYSET DRIVEN cursor, when the cursor is opened a (new) table is created containing pointers to the rows satisfying the specified predicate. FETCH statements use the pointers to access the result set.

   (e) Why deferred constraint checking is preferable in most transaction processing applications.

   Solution: Early statements in a transactions might make some constraints (temporarily) false, and then later statements might make them true again. Immediate constraint checking would make the transaction abort.

   (f) What are the advantages of stored procedures

   Solution:

   Among the advantages are:

   - They need not be prepared every time they are used
   - Less information need be passed between the application and the DBMS
   - Authorization can be at the procedure level
   - Security is improved
   - The application need now know the schema of the database
   - Maintenance is simplified

   (g) How can the same JDBC program be used to access different DBMSs

   Solution:

   The same program can use different drivers for different DBMSs

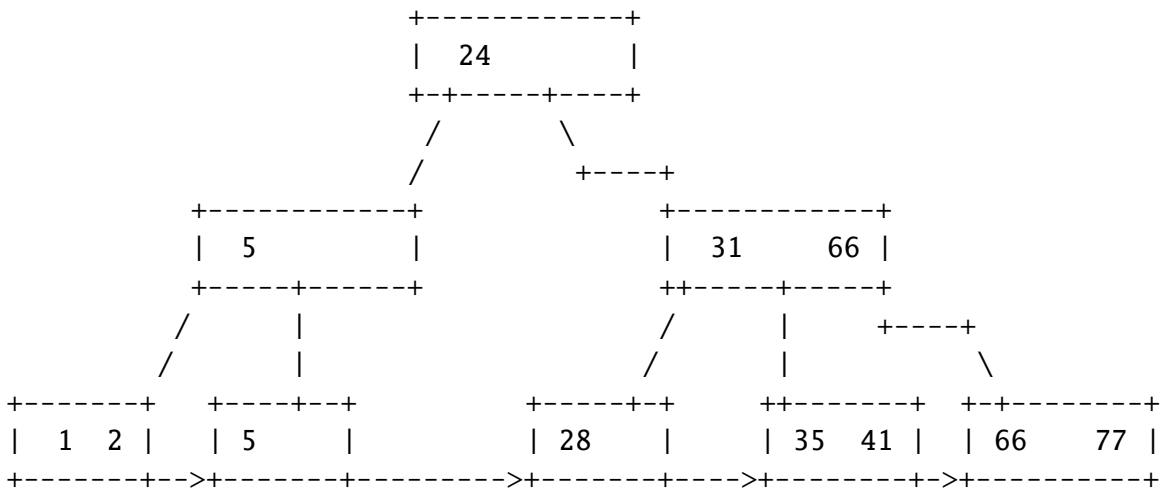
Problems for Chapter 9: Physical Data Organization and Indexing

1. Propose an algorithm for finding an intersection of two large relations (*i.e.*, ones that do not fit in main memory) using hashing.

   Solution:

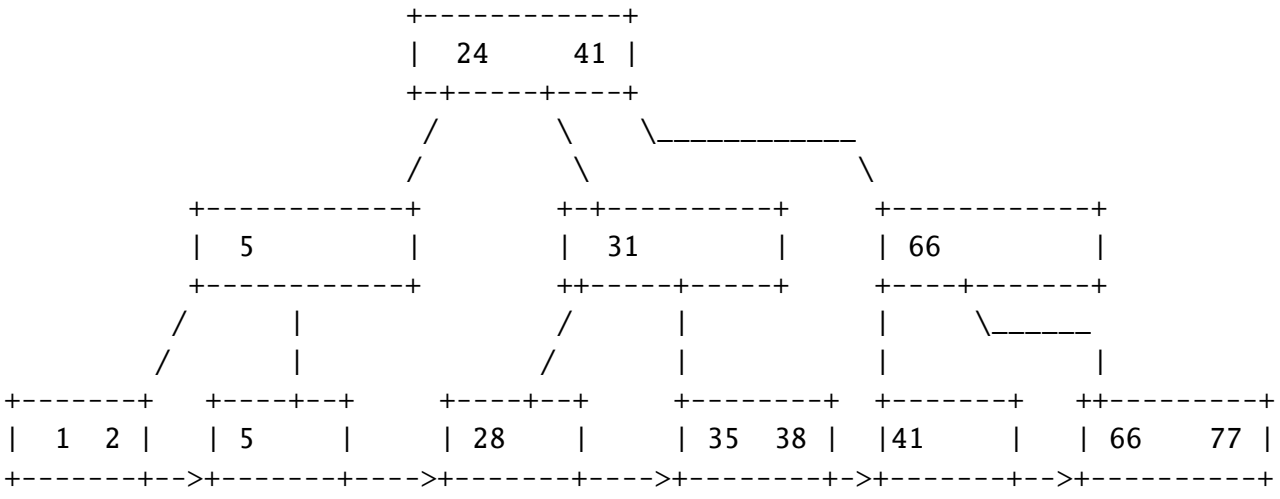
   Hash each relation on all attributes using the same hash function. Keep the distinction between the two parts of the bucket, which came from different relations. Tuples that fall into the intersection must fall into the same bucket. If the bucket is small enough, do the intersection in main memory. Else hash the tuples in the bucket again using a different hash function. It is reasonable to expect that the second-level buckets will fit into the main memory and then intersection within each bucket can be done in main memory.

2. Consider the following B+-tree, where a node can contain two search key values and three pointers. Show the B+-tree after inserting a new record with search key value 38. (You must redraw the tree when giving the answer– leave the original tree intact.)



Solution:

The insertion will split the node 35,41, which will result in a split of the mid-level node 31,66 and filling up of the root node.



3. Assume that you have a tree **clustered** index on a table of 100,000 tuples where

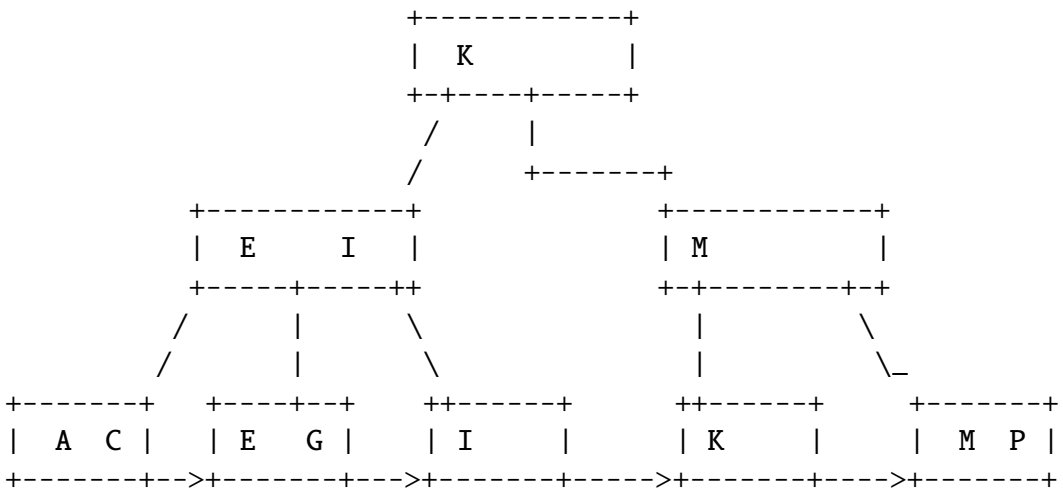
   - A page can hold 20 data tuples
   - An index page can have up to 100 pointers

   What is the *minimum* possible number of levels such an index can have? Draw a sketch of such an index and of the table that it indexes.

   Solution:

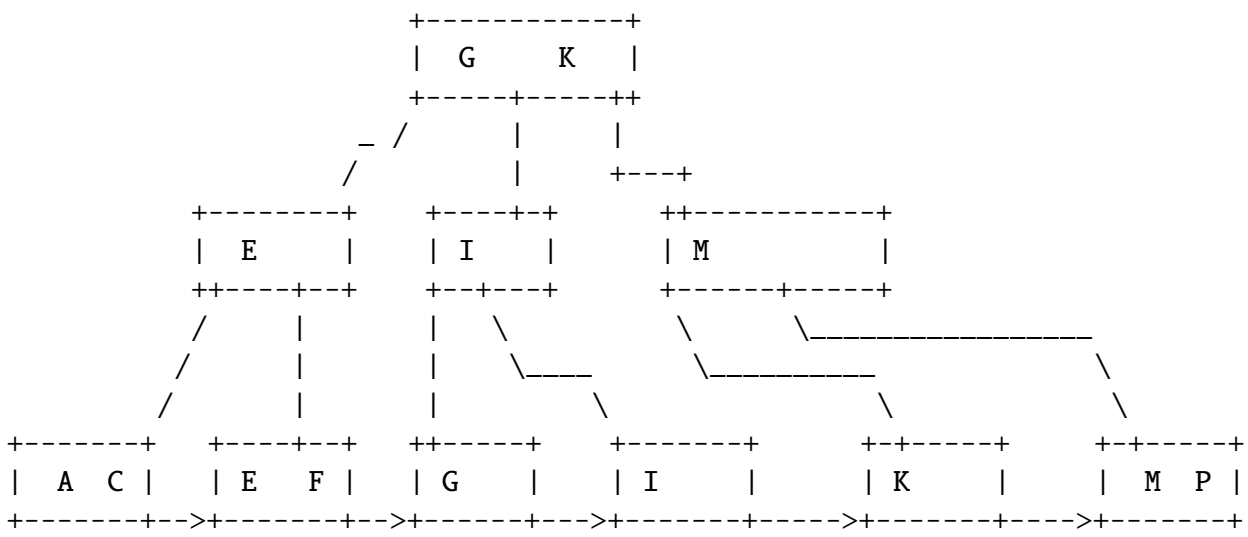
   The table has 100000/20 = 5000 pages. Since the index is clustered, the index can be sparse with 1 pointer per page, so it needs 5000 pointers. The minimum number of levels will result if the nodes of the tree are maximally packed. Thus, the minimum depth is $ceiling(log_{100}5000) = 2$.

4. Consider the following B+-tree, where a node can contain two search key values and three pointers. Show the B+-tree after inserting a new record with search key value F. (Please redraw the tree.)



Solution:

The insertion splits the node EG, which in turn splits the mid-level node EI. (We assume that spillover to siblings is not used).



5. Assume you have a table that contains 27,000 records, and you have an *unclustered* B^+-tree index on the table. Assume further that each index node can contain up to 100 pointers.

   (a) What is the **maximum** possible number of levels in the B^+-tree index for this table? Draw a sketch of this "maximal height" B^+-tree for the 27,000 record table.

   Solution:

   Since the index is unclustered, it requires one pointer per record in the data file. The tree will have the maximal depth is the nodes of the tree are minimally packed. This means that the root has only 2 pointers and the rest of the nodes have 50 pointers. So, the maximal depth will be $1 + ceiling(log_{50}13500) = 4$

   (b) What is the **minimum** possible number of levels in the B+-tree index for this table? Draw a sketch of this "minimal height" B+-tree for the 27,000 record table.

   Solution:

   Again, the index needs 27,000 pointers. The tree has minimal depth if the nodes are maximally packed. Therefore, the min depth is $ceiling(log_{100}27000) = 3$.

6. Assume that the rows of a table, R, are stored in an unsorted heap file, F, having M pages.

   (a) If no index is available, how many I/O operations on average will be required to compute the result of a SELECT statement with WHERE condition $a = 100$, where a is an attribute of R?

   Solution

   If a is a key then at most one row satisfies condition. Thus if the row is in R then M/2 and if not then M. If a is not a key then any number of rows might satisfy the condition, hence all of F must be scanned and M I/O operations are required.

   (b) If a secondary static hash index on attribute a is now added for accessing F, how would you calculate the cost of the query in (6a)?

   Solution

   The cost of the query is equal to the cost of finding the index entry in the overflow chain of bucket (which depends on how long the chain is and whether a is a key) plus one additional I/O operation for getting the row from F.

   (c) What is a clustered index? Is the index of (6b) clustered?

   Solution:

   The rows in the data file are ordered in same way as index entries in index file. The index of (6b) not clustered.

   (d) Extendable hashing uses the concepts of global depth and local depth.

      i. Which of these two is related to the size of the directory?
      Solution - global

      ii. All entries in a bucket agree on the last (local or global?) depth bits of the result of the hash.
      Solution - local

      iii. If a bucket overflows, the directory might have to be expanded. What is the condition on local and global depth that determines whether expansion is required?
      Solution - global depth = local depth of bucket that overflowed

      iv. How would you calculate the I/O cost of the query in (6a) assuming an extendable hash index on attribute a has been created for F
      Solution - Assuming the directory is not in memory: one I/O to get the directory page + one I/O to get the index entry from the bucket + one I/O to get the row from F

7. Figure 7a shows a portion of a B^+ tree whose search key is student name and in which each page can contain at most two entries.

   (a) What does the tree look like after insertion of *vera*?

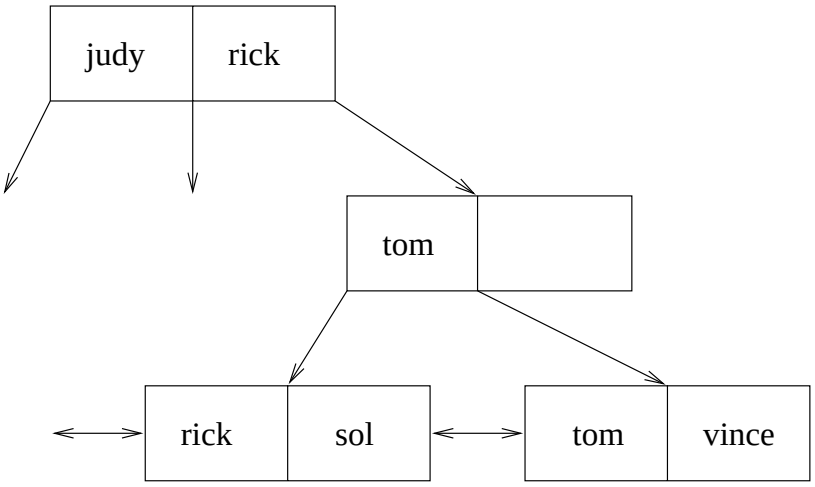


Figure 1: Initial state of B^+ tree.

Solution:
See Figure 2.

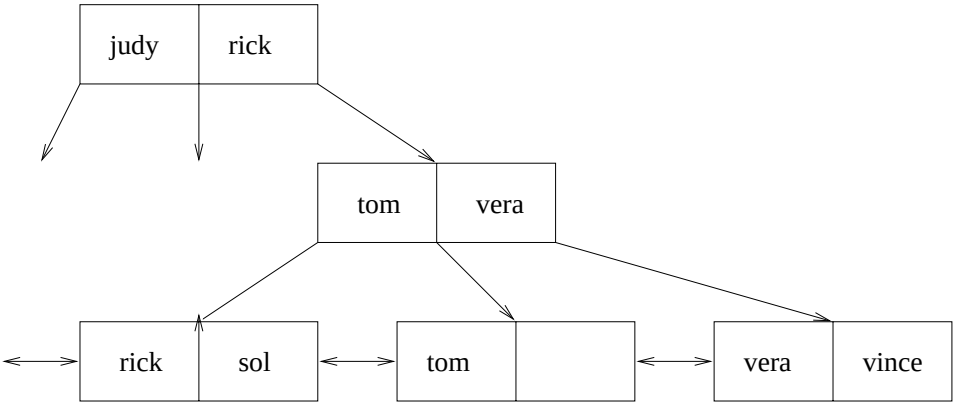


Figure 2: State of B^+ tree after the insertion of *vera.*

   (b) What does the tree look like if *rob* is inserted (after *vera* has been inserted)?
   Solution: See Figure 3.

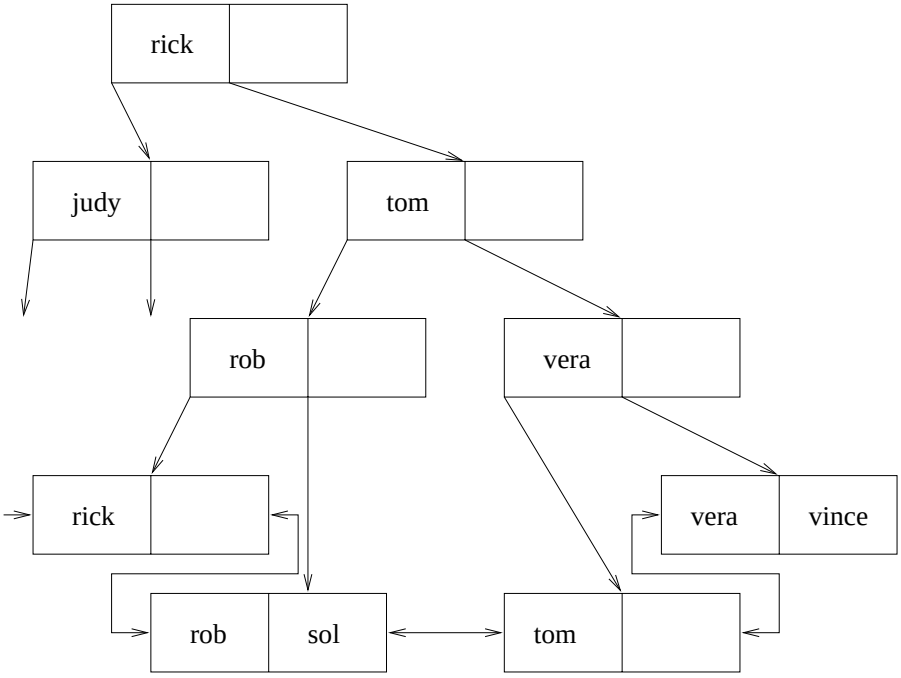


Figure 3: State of B^+-tree after the insertion of *rob*.

8. A table has 100,000 rows and each row occupies 200 bytes. The table is stored on a disk which has 4^k bytes per page. Compute the maximum (worst case) cost (number of I/O operations) of doing an equality search on the primary key assuming the following access paths. Make your reasoning clear - a number without an explanation gets no credit.

   (a) The data file is unsorted and has no index.

   Solution

   The data file has $(10^5 * 200)/(4 * 10^3)$ pages = 5000 pages
   In the worst case the entire file has to be searched costing 5000 I/O operations.

   (b) The data file is sorted on the primary key and has no index.

   Solution

   Use binary search. Cost = $log_2 5000$. Hence worst case is 13 I/O operations.

   (c) There is an unclustered static hash index whose search key is the primary key. Assume all buckets are stored on disk and that each bucket has one overflow page.

   Solution

   Cost of hash is 0; cost of searching bucket = 2 I/O operations; cost of retrieving row = 1 I/O operation; total = 3 I/O operations.

   (d) There is an unclustered B^+ tree index whose search key is the primary key. Assume that each entry in the tree occupies 20 bytes and the entire tree resides on the disk.

   Solution

   $(4 * 10^3)/20$ = 200 maximum entries per page; 100 minimum entries per page (worst case)
   $10^5/10^2$ = 1000 leaf pages
   Rounding $log_{100} 1000$ up to the next integer (2) we get that the number of levels of the tree is 3. Hence 3 I/O operations are necessary to search the tree and an additional I/O operation is necessary to fetch the row.

9. Assume a table has 100,000 rows and each row occupies 100 bytes. Disk pages are 4000 bytes and the worst case time to access a page is 20ms. Estimate the worst case time of doing an equality search on a candidate key under the following conditions.

   (a) An (unsorted) heap file with no index.

   Solution:

   size of file = $(10^5 * 10^2)/(4 * 10^3) = 2500$ pages
   worst case time = $2500 * 20$ms = 50 sec

   (b) An unclustered B^+ tree in which each entry occupies 20 bytes.

   Solution:

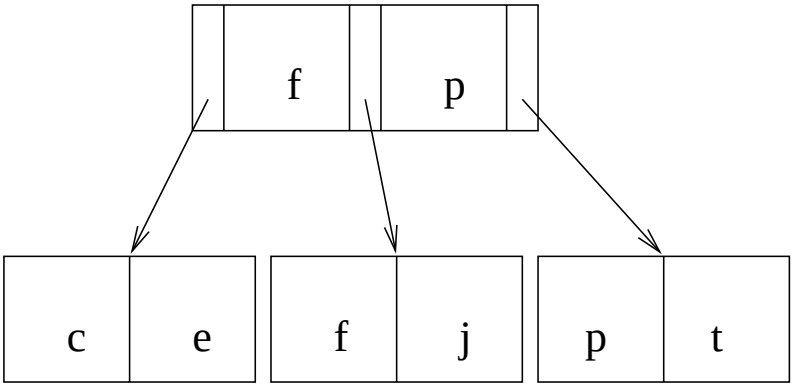
   max. number of entries in a page = $(4 * 10^3)/20 = 200$
   min. number of entries in a page = 100 (worst case)
   tree must have 10^5 entries, hence a 3 level tree is required
   3 I/O operations to search tree + 1 I/O operation to fetch row = 4 I/O ops
   $4 * 20$ms per I/O operation = 80 ms

   (c) A clustered B^+ tree.

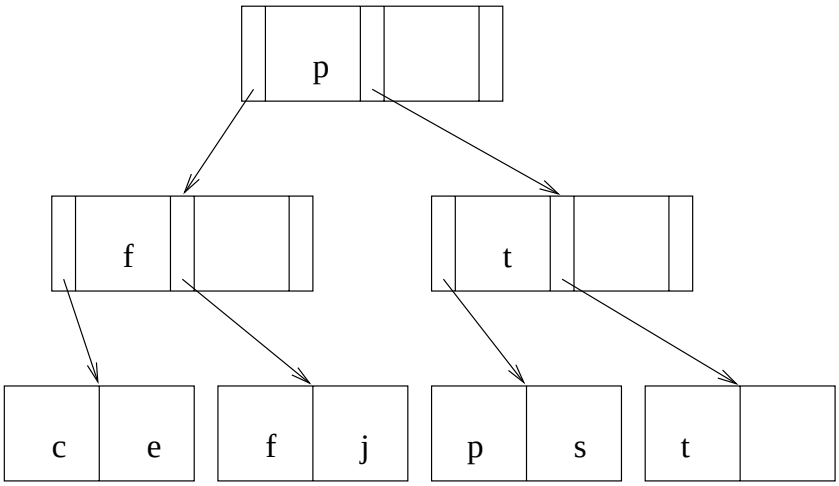
   Solution:

   100 separator entries per page (worst case)
   4000/100 = 40 rows per page max at leaf level
   20 rows per page at leaf level (worst case)
   leaf level has 100,000/20 = 5000 pages (worst case)
   number of separator levels in tree is smallest integer k satisfying $100^k \geq 5000$
   hence $k = 2$
   total number I/O operations = 2 (to search separator levels) + 1 (to get index page)
   cost = 60 ms

10. Search key value *s* is to be inserted in the following B^+ tree. Show what the tree looks like after the insertion.



Solution:



11. Pages in a database have P bytes. A table T has R rows and n integer attributes, A1, A2, .An. Each attribute is r bytes long. Assume that r is much greater than the size of a pointer, so the pointer size can be ignored when computing the size of an entry in an index. Assume that the domain of A1 is the integers between 1 and 100 and that the values of this attribute are randomly distributed in the domain. Consider the following statement:

```
SELECT A2
FROM   T
WHERE A1 <= 50  AND A1 > 25
```

Evaluate the cost, in terms of the number of I/O operations, of executing the statement under the following conditions. In order to get credit you must explain your work clearly.

(a) The table is stored as a heap (no indexes).

Solution:

scan entire table
size of table = $n * r * R$
number pages in table = $(n * r * R)/P$ = cost

(b) There is a non-integrated (separate index file) clustered index with search key A1.

Solution:

Search down the tree using A1 = 25, follow pointer to first qualifying row in data file, scan data file and return all rows until row with A1>50 is found.
fan-out of a page in the index = P/r (or P/2r for a worst case analysis) num. of levels in tree = $\log_{P/r} R$ = cost of tree search
cost of file scan = $.25((n * r * R)/P)$ since approximately a quarter of the rows have to be returned (values of A1 are randomly distributed)
total cost = $\log_{P/r} R + 1 + .25((n * r * R)/P)$

(c) There is an unclustered index with search key (A2, A1).

Solution:

Scan leaf level of tree (index covering). An index entry has 2r bytes, hence the (minimum) number of leaf level pages in the index = R/(P/2r) (worst case analysis doubles the cost).

Problems for Chapter 10: The Basics of Query Processing

1. Consider the join $\mathbf{r} \bowtie_{\texttt{A<B}} \mathbf{s}$ and the following join computation techniques:

   (a) Sort-merge

   (b) Hash-based

   (c) Index-nested loops

   Briefly (in one paragraph per technique) explain which technique is *reasonably* good in the above case. (In case of at least one technique you have to distinguish between two cases.) The explanations should be short and to the point.

   Solution:

   Sort-merge join is a reasonable technique. In the last phase of sorting, we can perform the join by comparing the attribute `A` of the front tuples of **r** with the attribute of the front tuple of **s** and select those where the condition `A<B` is satisfied.

   Hash-based join is useless in this case, since the join condition does not involve equality.

   Index-nested loops join will work reasonably well provided that the index on `B` in **s** is clustered. For unclustered index, this technique will result in one I/O per matching tuple, which is worse than plain scan.

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

   - 5,000 tuples with 10 tuples per page
   - A 2-level B^+ tree index on attribute *A* with up to 100 index entries per page
   - Attribute A is a candidate key of **s**
   - The values that the attribute A takes in relation **s** are uniformly distributed in the range 1 to 100,000.

   (a) Assuming that the aforesaid index on A is *unclustered*, estimate the number of disk accesses needed to compute the range query $\sigma_{A>1000 \wedge A<6000}(\mathbf{s})$.

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

   Explain your reasoning.

   Solution:

   (a) *Unclustered index*: Since the range 1000–6000 is a fraction 5000/100000 = 1/20 or the total range, it means that the selection will roughly select 1/20*5000 = 250 tuples. Since the index is unclustered, it will take 1 I/O per data entry to fetch it through the index.

   Searching the index itself will take 2 + the number of pages that contain the relevant index entries (ceiling(250/100)=3). Therefore, the total number of disk accesses will be 2+3+250=255.

   (b) *Clustered index:* In this case, we only need to find the first data page at the cost of 1 index search, *i.e.*, 2. All the data records of interest will be grouped in maximum 250/10 +1 = 26 pages (1 extra if the first data entry of interest starts in the middle of a page). So, the total number of accesses is 2+26=28.

3. Consider the relation STUDENT(Id,Major,Status), which has:

   - A clustered hash index on Major and no other indices.
   - 100 pages of data spread over 50 majors.
   - The domain of Status has 9 values.

   Find the best access path to evaluate the expression

   $\sigma_{\texttt{Major='CS' OR Status='U3'}}$ (STUDENT)

   and estimate the cost of using this access path.

   Solution:

   Since we have a disjunction, we need to evaluate $\sigma_{\texttt{Status='U3'}}$(STUDENT) anyway. This selection requires a scan of the relation at the cost of 100 pages plus the cost of writing out the result.

4. You are given the following information about two tables, r and s.

   - Table r occupies 800 pages, 20 rows per page, one of its attributes is A
   - Table s occupies 200 pages, 10 rows per page, one of its attributes is B
   - Main memory has 52 buffers

   (a) Compute the minimum cost (measured as the number of I/O operations) of a block-nested loops join $\texttt{r} \bowtie_{\texttt{A=B}} \texttt{s}$. Specify how the buffers are used and explain your reasoning carefully. An answer that is just a number gets no credit.

   Solution:

   50 input buffers for s, one input buffer for r, one output buffer
   s is broken into 4 segments, one pass over r for each segment
   total cost = 4 passes over r (= 3200 I/O ops) + 1 pass over s (= 200 I/O ops) = 3400 I/O operations

   (b) Assume there is an unclustered, 3-level, B+ tree index on r with search key A and that each row of s joins with 5 rows of r. Compute the cost of an index-nested loops join. Explain your reasoning carefully. An answer that is just a number gets no credit.

   Solution:

   200 I/O ops to scan S
   num. rows in s = 2000
   for each row of s it takes 3 + 5 I/O ops to get the 5 matching rows of r
   total cost = $2000 * 8 + 200$ = 16,200 I/O ops.

Problems for Chapter 11: An Overview of Query Optimization

1. Consider the following relational schema:

   EMPLOYEE(Id,Name)
   WORKSIN(EmpId,DeptName)
   DEPARTMENT(DeptName,Building)

   Assume that keys are underlined and:

   - The relation EMPLOYEE is sorted on the Id attribute.
   - In DEPARTMENT: *Unclustered* index on DeptName; *clustered* on Building. No other indices.
   - Dozens of departments can reside in the same building.

   (a) Write an SQL query that generates a table of employee names who work for department "Sales" in building E213.

   Solution:

```
SELECT E.Name
FROM  EMPLOYEE E,  WORKSIN W,  DEPARTMENT D
WHERE E.Id = W.EmpId  AND W.DeptName = D.DeptName
        AND D.Building = 'E213'  AND D.DeptName = 'Sales'
```

   (b) Give the "naive" translation of your SQL query into the **relational algebra** (as given by the general translation of SQL to relational algebra.)

   Solution:

$$\pi_{\text{Name}}(\sigma_{\text{Id=EmpId AND WorksIn.DeptName=Department.DeptName AND Building='E213' AND Department.Name='Sales'}}(\text{EMPLOYEE} \times \text{WORKSIN} \times \text{DEPARTMENT}))$$

(c) Describe **carefully and completely** how you would actually most efficiently evaluate the above query. That is, list the precise sequence of relational operations to be performed, the relations on which these operations are to be performed, **and** the methods to be used to compute the result of each operation. (As part of the solution you will need to decide which index to use and why.)

Solution:

Since there are no indices on WORKSIN and this relation is potentially large, we should try to reduce DEPARTMENT as much as possible and as cheaply as possible. Since `DeptName` is a key in DEPARTMENT and we have an index on this attribute, we can apply the selection $\sigma_{\texttt{DeptName='Sales'}}(\text{DEPARTMENT})$ first. This is supposed to produce a single tuple. Selection $\sigma_{\texttt{Building='E213'}}$ is applied next. It will either leave the selected tuple intact or will eliminate it completely (in the latter case the query will return empty set).

The next step is to join the result with WORKSIN. (Since we are joining WORKSIN with a single tuple, this operation is essentially a selection.) We can use block-nested loops join here, which can be computed in 1 scan of WORKSIN. Finally, we can join the result with EMPLOYEE. Since EMPLOYEE is sorted on `Id` and what is left of WORKSIN is presumably small (since this result contains only the employees from one department out of many dozens), it is probably best to use sort-merge join here.

(d) Give the relational algebra expression for this query that most closely corresponds to the way you chose (in the previous subproblem) to efficiently evaluate this query.

Solution:

$\pi_{\texttt{Name}}((\sigma_{\texttt{Building='E213'}}(\sigma_{\texttt{DeptName='Sales'}}(\text{DEPARTMENT})) \bowtie \text{WORKSIN}) \bowtie_{\texttt{EmpId=Id}} \text{EMPLOYEE})$

2. Consider the following database schema:

   STUDENT(StudID,SName,SAddr)
   TRANSCRIPT(StudID,DeptCode,CrsNum,Semester,Grade)

   (a) Write an SQL query that generates a table of the SNames of all students taking courses in the department with DeptCode "CS".

   Solution:

```
SELECT S.SName
FROM STUDENT S, TRANSCRIPT T
WHERE S.StudID = T.StudID AND DeptCode = 'CS'
```

   (b) Give the "naive" translation of your SQL query into the **relational algebra** (as given by the general translation of SQL to relational algebra.)

   Solution:

   $\pi_{\texttt{SName}}(\sigma_{\texttt{Student.StudId=Transcript.StudId AND DeptCode='CSE'}}(\text{STUDENT} \times \text{TRANSCRIPT}))$

   (c) Assume that there is an index with search key DeptCode on the transcript table, and **no** other indexes. Describe **carefully and completely** how you would evaluate the above query in a most efficient way. Give a precise query plan and carefully describing each operation.

   Solution:

   First select TRANSCRIPT on DeptCode, then project out everything but StudId, and then join the result with STUDENT. Use sort-merge join or block-nested loops. Before the join, get rid of Address in STUDENT to reduce size. Finally, project out everything but SName.

   (d) Give the relational algebra expression for this query that most closely corresponds to the way you chose (in the previous subproblem) to efficiently evaluate this query.

   Solution:

   $\pi_{\texttt{SName}}(\pi_{\texttt{StudId}}(\sigma_{\texttt{DeptCode='CS'}}(\text{TRANSCRIPT})) \bowtie \pi_{\texttt{StudId,SName}}(\text{STUDENT}))$

3. Describe an intuitively efficient query evaluation plan for the following query, where the relations involved have the following schemas: REL1(A,B), REL2(B,C).

```
SELECT T.A
FROM  REL1 T
WHERE  EXISTS (
                    ( SELECT R.B
                      FROM  REL1 R
                      WHERE R.A = T.A)
                     EXCEPT
                    ( SELECT O.B
                      FROM  REL2 O
                      WHERE O.C = 123)
                   )
```

Solution:

(a) Select REL2 on C and project on B. Denote the result by REL3.

(b) Sort REL3 and REL1 on B.

(c) In a merge-style algorithm remove tuples from REL1 that join with REL3 on attribute B. Let REL4 denote the result.

(d) Project REL4 on the attribute A.

4. Consider the following schema, where the keys are underlined:

   EMPLOYEE(SSN, Name,Dept)
   PROJECT(SSN,PID,Name,Budget)

   The SSN attribute in PROJECT is the Id of the employee working on the project and PID is the Id of the project. There can be several employees per project, but the functional dependency PID → Name,Budget holds (so the relation is not normalized). Consider the query

```
SELECT    P.Budget, P.Name, E.Name
FROM      EMPLOYEE E, PROJECT P
WHERE     E.SSN = P.SSN AND
          P.Budget > 99 AND
          E.Name = 'John'
ORDER BY  P.Budget
```

   Assume the following statistical information:

   - 10,000 tuples in EMPLOYEE relation
   - 20,000 tuples in PROJECT relation
   - 40 tuples/page in each relation
   - 10-page buffer
   - 1,000 different values for E.Name
   - The domain of Budget consists of integers in the range of 1 to 100
   - Indices:
     - EMPLOYEE relation:
       On Name: Unclustered, hash
     - PROJECT relation:
       On SSN: Unclustered, hash
       On Budget: Clustered, 2-level B^+ tree

   Question: Find the best execution plan, draw it, and estimate its cost.

   Solution:

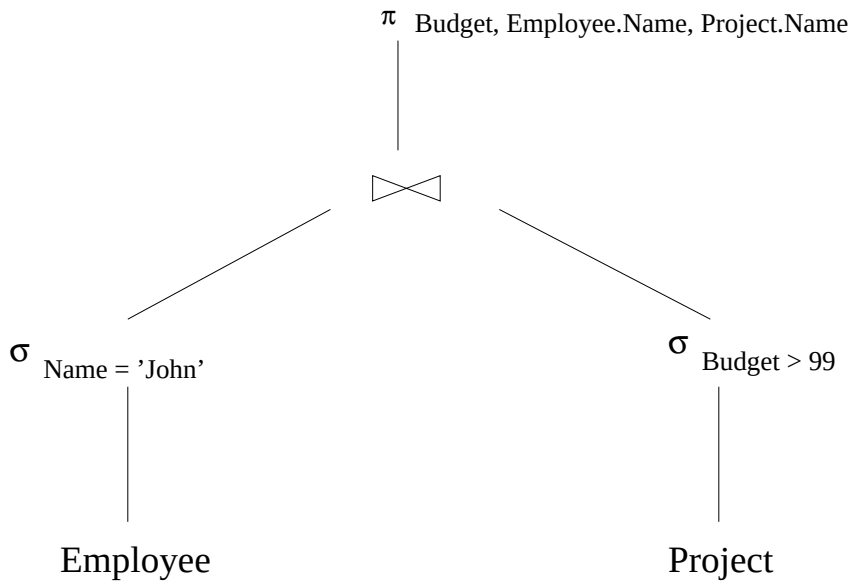
   There are three possible candidates to consider: (1) push both selections; (2) push selection to EMPLOYEE only; (3) push selection to PROJECT only. Option (1) yields the least number of I/Os.

   Selection $\sigma_{\text{Name='John'}}$(EMPLOYEE): Since Name can have 1000 values and assuming they are uniformly distributed, the result will have 10000/1000 = 10 tuples. Since the index on Name is unclustered hash, this takes 1.2+10 = 12 I/Os.

   Selection $\sigma_{\text{Budget}>99}$(PROJECT): Since the range of Budget is 100, this reduces the size by 100. That is, 20000/100 = 200 tuples, 5 pages. Since the index is a 2-level clustered B^+ tree, this will take 2 + "the number of result pages" I/Os. The result fits in 5 pages, but the first tuple may start in the middle of a page. So, we may have to fetch 6 pages. Thus, the cost is 2+6=8 pages.

Since we have a 10 page buffer, we can keep both selections in main memory and perform the join in main memory. Therefore, the final cost is 12+8=20 I/Os.

The query plan is shown below.



5. Consider the following schema, where the keys are underlined:

   COMPANY(Name, Industry, RegistrationState)
   CITY(Name, State, Population)

   COMPANY is an unnormalized relation that contains information on the states where companies are registered (Name → RegistrationState is an FD) and the industries where they belong (company-industry is a many-to-many relationship). The CITY relation contains information about the number of people living in each city. Different cities can have the same name in different states, but city names within the same state are unique. Consider the following query:

```
SELECT   C.Name, T.Name
FROM     COMPANY C, CITY T
WHERE    C.RegistrationState = T.State AND
          T.Population > 9000000 AND
          C.Industry = 'Apparel'
ORDER BY     C.Name
```

   Assume the following statistical information:

   - 20,000 tuples in COMPANY relation
   - 5,000 tuples in CITY relation
   - 50 tuples/page in each relation
   - 6-page buffer
   - The domain of Population consists of integers in the range of 1000 to 10,000,000
   - There are 50 industries represented in the database and 50 states.
   - Indices:
     - COMPANY relation:
       On Industry: Clustered, hash
     - CITY relation:
       On State: Clustered, 2-level B^+ tree

   Question: Find the best execution plan for the above query, draw it, and estimate its cost.

   Solution:

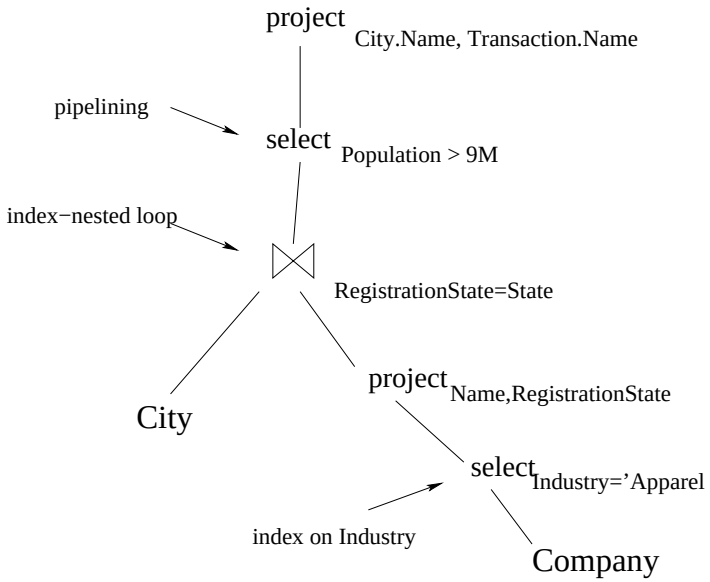
   The index on COMPANY cannot be used in a join, so it makes sense to push the selection and the projection to COMPANY. The size of $\pi_{\text{Name,RegistrationState}}(\sigma_{\text{Industry=Apparel}}(\text{COMPANY}))$ is: 20000/50 = 400 tuples (after selection) = 8 pages. Projection further reduces the size by 2/3 to 5.4 (*i.e.*, 6) pages.

   If we use the clustered index on City.State in the join, we would have to use the index 400 times for each tuple in $\pi_{\text{Name,RegistrationState}}(\sigma_{\text{Industry=Apparel}}(\text{COMPANY}))$. But if we do it smartly only for the states that actually occur in that expression, we still would have to use the index 50 times. This is cheaper than computing $\sigma_{\text{Population>9M}}(\text{CITY})$, because this selection will need to scan CITY (100 pages).

Since CITY is clustered on `State`, this relation is ordered on the `State` attribute. Therefore, we can pull $\pi_{\texttt{Name,RegistrationState}}(\sigma_{\texttt{Industry=Apparel}}(\text{COMPANY}))$ through the 6-page buffer using just one page and join it with CITY using the index on CITY.`State`.

Thus, the total cost is: 8 + 50 = 58 I/Os.

The query plan is shown below.



6. A company sells merchandise through agents. Each sales agent is assigned one or more cities to cover. Consider the following schema, where the keys are underlined:

   AGENT(Name,City)
   LOCATION(City, ZIP)
   TRANSACTION(Date, ZIP, Item)

   ($\langle d, z, i \rangle \in$ TRANSACTION means that item i was sold on date d in the area with zip code z.)

   Consider the following query:

$$\pi_{\texttt{City,Date}}(\sigma_{\texttt{Name='007' AND Item='land'}}(\text{AGENT} \bowtie \text{LOCATION} \bowtie \text{TRANSACTION})$$

   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.

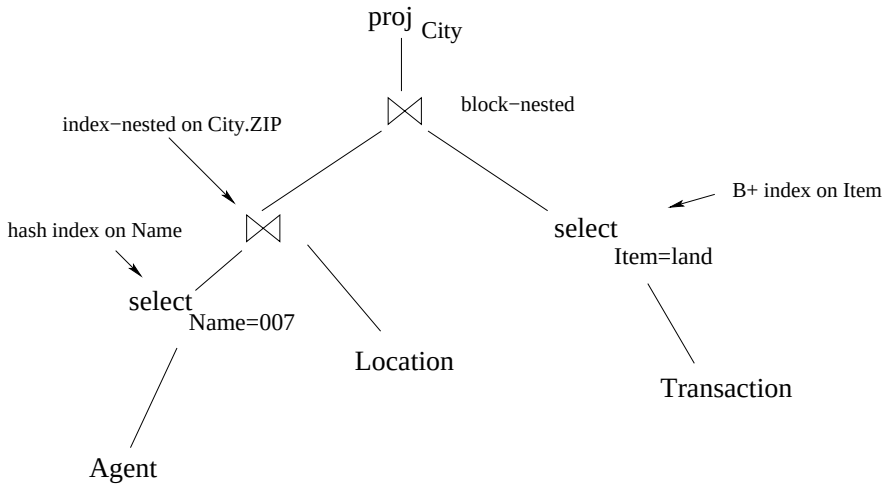
   Assume 50 buffer pages and the following statistics and indices:

   - AGENT: 1,000 tuples, 5 tuples/page; 200 agents.
     Clustered hash index on Name.
     Unclustered B$^+$ tree index on City.
   - LOCATION: 2500 tuples, 50 tuples/page; 500 cities.
     Unclustered hash index on ZIP.
   - TRANSACTION: 10,000 tuples, 20 items; 10 tuples/page.
     2-level clustered B$^+$ tree index on Item.

   Solution:

   One most promising expression is where the selection Name='007' is pushed, the other is where Item='land' is pushed, and the third is where both are pushed.

   The most efficient query plan is depicted in the figure.



   Sizes:

- $\sigma_{\texttt{Name='007'}}$(AGENT): 5 tuples, 1 page, 1 I/O (since `Name` is clustered).
- $\sigma_{\texttt{Item='land'}}$(TRANSACTION): 500 tuples, 50 pages. There can be up to 250 ZIP codes among these tuples, 50 I/Os + 2 to search the index.
- LOCATION: 50 pages

The first join can potentially create 25 tuples (5 ZIP codes per city, 5 cities selected from Agent) at the cost of 25 I/O. The result will occupy at most 10 pages (given that tuples can be twice as long).

In the block-nested loops join, we will scan the selection on TRANSACTION once in the outer loop (50 pages) and the result of the previous join (10 pages) at most twice in the inner loop. (Twice because we have 50 buffer pages only and the outer relation is also 50 pages long). So, the cost of this join is 50+20=70pages.

Total: 1 + 50 + 2 + 25 + 70 = 148 page I/Os.

7. (a) The equivalence

$$\pi_{a}(R \bowtie_{c} S) \equiv \pi_{a1}(R) \bowtie_{c} \pi_{a2}(S)$$

where $a1$ and $a2$ are disjoint subsets of a satisfying $a = a1 \bigcup a2$ holds under certain conditions on $a1$, $a2$, and c. State the conditions.

Solution:

$a1$ and $a2$ are subsets of a corresponding to R and S and all attributes named in c are contained in $a1$ and $a2$

(b) Using the schema:

```
Sailors(sid: integer, sname: string, rating: integer, age: real)
        key - sid
Boats(bid: integer, bname: string, color: string)
        key - bid
Reserves(sid: integer, bid: integer, day: date)
        key - sid, bid, day
```

and assuming that Sailors has 100 pages and Reserves has 200 pages, how many I/O operations must be performed to execute a natural join using a page-at-a-time nested-loops join algorithm?

Solution:

For each page of Sailors make a complete pass of Reserves.
100 passes over Reserves (= 100*200 page I/O's) +
1 pass over Sailors (= 100 page I/Os) = 20,100
Alternatively, 20,200 if Sailors and Reserves are reversed

(c) Consider the SELECT statement:

```
SELECT S.sname, S.rating
FROM Sailors S, Reserves R
WHERE S.sid = R.sid AND R.day = '5/11/98'
```

which uses the schema of (7b). Express it as an unoptimized relational algebra expression using the operators natural join, select, and project.

Solution:

$$\pi_{sname,\ rating}\ (\sigma_{day='5/11/98'}\ (S \bowtie R))$$

(d) Give an equivalent relational algebra expression for the same query that can be evaluated more efficiently. Explain why you expect more efficient evaluation (no numbers).

Solution:

$$\pi_{sname,\ rating}\ (\sigma_{day='5/11/98'}\ (R) \bowtie S)$$

This eliminates irrelevant rows of R before taking the join and thus reduces the number of tuples produced by the join and its cost. Alternatively,

$$\pi_{sname,\ rating}\ (\sigma_{day='5/11/98'}\ (\pi_{day,\ sid}\ (R) \bowtie \pi_{sname,\ rating,\ sid}\ (S)$$

Further efficiency since early projection reduces row size.

8. You are given the following tables:

Student(StudId, Name, Addr, Status)
Transcript(Id, CrsCode, Semester, Grade)

(a) Write a SELECT statement that outputs the names of all students who took CSE305 in the spring of 2000.

Solution:

```
SELECT S.Name
FROM STUDENT S, TRANSCRIPT T
WHERE T.Id = S.StudId AND T.CrsCode = 'CSE305'
             AND T.Semester = 'S2000'
```

(b) A naive query execution plan for your answer to (8a) would first execute the FROM clause, then apply the WHERE condition to the result, and then eliminate unwanted columns using the attributes named in the SELECT clause. Express this plan as a relational algebra expression.

Solution:

$$\pi_{Name}(\sigma_{Id=StudId AND CrsCode='CSE305' AND Semester='S2000'}(Student \times Transcript))$$

(c) We're interested in a more efficient execution plan for this query than your answer to (8b). Such a plan uses a join operation on relations which are as small as possible (some rows of the tables to be joined have been eliminated and the remaining rows have been shortened). Give a relational algebra expression equivalent to your answer to (8b) that can be evaluated more efficiently. The amount of credit you get depends on how efficient the expression is.

Solution:

Full credit for:

$$\pi_{Name}(\pi_{Name,StudId}(Student) \bowtie_{StudId=Id} (\sigma_{CrsCode='CSE305' AND Semester='S2000'}(\pi_{CrsCode,Semester,Id})(Transcript)))$$

or

$$\pi_{Name}(\pi_{Name,StudId}(Student) \bowtie_{StudId=Id} (\pi_{Id}(\sigma_{CrsCode='CSE305' AND Semester='S2000'}(Transcript))))$$

Problems for Chapter 12: Database Tuning

1. This problem uses the table EMPLOYEE (Id, DeptId, Name, Salary) with primary key Id.

   (a) Assume EMPLOYEE has a clustered index on Id that cannot be changed. It is proposed that the frequently processed query

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

   be handled using index covering. What kind of index would you propose to do this? Specify the search key attributes and the type (clustered/nonclustered, B^+ tree/hash).

   Solution:

   B^+ tree with search key (DeptId, Name), unclustered

   (b) If your answer to the previous part was a hash (B^+ tree) explain whether a B^+ tree (hash) would work as well.

   Solution:

   Hash would not work since the search key of the index would have to be (DeptId, Name) and Name is not known.

   (c) The application has the following frequently executed queries:

   ```
   (1) SELECT E.Id, E.DeptId
       FROM EMPLOYEE E
       WHERE E.Name = :name
   ```

   ```
   (2) SELECT AVG (Salary)
       FROM EMPLOYEE E
       WHERE E.DeptId = :dept
   ```

   Now assume that the storage structure can be changed, that there are only a few employees with the same name, and that departments have a large number of employees:

   i. What storage structure would you use for the table? If an index is involved state whether it is a B^+ tree or hash, clustered or unclustered.
   Solution:
   clustered B^+ tree (or hash) on DeptId

   ii. Describe all additional indexes (don't use index covering). For each state whether it is a B^+ tree or hash, clustered or unclustered.
   Solution:
   unclustered on primary key Id, could be B^+ tree or hash
   unclustered on Name, could be B^+ tree or hash

   iii. Describe the query plan (you hope) the DBMS would use for each query.
   Solution:
   for query (1) - Hash or search on Name. All index entries corresponding to Name are either in same bucket or clustered at leaf level. Follow pointers (there are only a few) to rows with given Name.

   for query (2) - Hash or search on DeptId. Scan the bucket or leaf level to get all rows of employees in the department and take avg. of salaries.

2. Consider the table PROFESSOR with attributes Id, Name, Dept, Salary, and Rank and primary key Id, and the statements:

```
    SELECT Name, Dept
(S1) FROM  PROFESSOR
    WHERE Id = :ssn
```

```
     SELECT Dept,  COUNT (*)
(S2)  FROM  PROFESSOR
      WHERE Salary > sal
      GROUP BY Dept
```

where ssn is a host variable and sal is a literal.

(a) Give an efficient query plan for S1 and state what index (search key, clustered, nonclustered, B^+ tree, hash) is used.

Solution:

clustered or unclustered, hash or B^+ tree on Id
use index to locate unique row satisfying Id = :ssn

(b) Now suppose, in addition to optimizing S1, we are equally concerned with optimizing the execution of S2. Assuming sal is a large number (most rows do not satisfy the WHERE clause) describe the query plan you would like the optimizer to use and any changes that might need to be made to your previous answer. State what index (search key, clustered, nonclustered, B^+ tree, hash) is used for each statement. Explain your reasoning.

Solution:

Use an unclustered B^+ tree index on Salary to get the rows satisfying the WHERE clause (only a few rows satisfy). Sort them by Dept and count size of each group. The answer to the previous part is unchanged.

or

Use a clustered B^+ tree on Salary. Scan leaf level starting from sal to get satisfying rows, sort on Dept, count size of each group. The answer to the previous part must be unclustered index on Id.

(c) Redo the previous part assuming that sal is a small number (most rows satisfy the WHERE clause). Explain your reasoning.

Solution:

Use clustered B^+ tree index on Dept. Scan entire leaf level, groups will be contiguous and number of members with appropriate salary of each group can be counted. Since most rows satisfy, most of the pages will have to be read anyway and a large sort will be necessary if rows are not already sorted on Dept. By clustering on Dept the sort is avoided and fast scanning techniques can be employed. The answer to the first part must be an unclustered index on Id.

3. (a) The table Employee(Id, DeptId, Name, Salary) has a clustered index with primary key Id and we cannot change this storage structure. It is proposed that the frequently processed query

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

be handled using index covering. What kind of index would you propose for doing this? Specify the search key attributes and the type (clustered/nonclustered, B^+ tree/hash).

Solution

Unclustered B^+ tree index with search key (DeptId, Name)

(b) If your answer to the previous part was a hash (B^+ tree) explain whether a B^+ tree (hash) would work as well.

Solution

Hash would not work since search key would have to be (DeptId, Name) and Name is not known.

(c) The application has the following frequently executed queries:

```
SELECT E.Id, E.DeptId          SELECT AVG (Salary)
FROM Employee E                FROM Employee E
WHERE E.Name = :name           WHERE E.DeptId = :dept
   (1)                            (2)
```

Now assume that the storage structure can be changed, that there are only a few employees with the same name, and that departments have a large number of employees. (Don't use index covering for this part of the problem.)

i. What storage structure would you use for the table? If an index is involved state whether it is a B^+ tree or hash, clustered or unclustered.
Solution
clustered B^+ tree or hash on DeptId

ii. Describe all additional indexes. For each state whether it is a B^+ tree or hash, clustered or unclustered.
Solution
Unclustered on primary key Id, could be B^+ tree or hash;
Unclustered on Name, could be B^+ tree or hash

iii. Describe the query plan (you hope) the DBMS would use for each query.
Solution] on (1); Hash or search on Name; all index entries corresponding to Name are either in same bucket or clustered at leaf level; follow pointers (there are only a few) to rows with given Name
on (2): Hash or search on DeptId; scan the bucket or leaf level to get all rows of employees in the department and take the average of their salaries

4. Consider the schema

```
Employee(Id: INTEGER, Name: STRING, Deptid: STRING, Salary: INTEGER)
Department(DeptId: INTEGER, Name: STRING)
```

Propose indexes appropriate for evaluating the query

```
SELECT D.Name, E.Name
FROM Employee E, Department D
WHERE E.DeptId = D.DeptId  AND E.Salary > :sal
```

(a) when it is expected that sal will be a large number.

Solution:

With sal a large number, only a few rows of E will qualify. Hence an efficient query plan would be one that uses a B^+ tree index on EMPLOYEE with search key Salary to fetch those rows in the outer loop of an index-nested loop join. For each row, the corresponding row of DEPARTMENT can be fetched in the inner loop using an index with search key DeptId.

(b) when it is expected that sal will be a small number.

Solution:

Since a large result set is expected an index-nested loop join will generally be inefficient. A sort-merge join will be better, especially if the tables are pre-sorted by the presence of clustered B^+ tree indexes on the join attributes: D.DeptId and E.DeptId.

5. Consider the schema

```
Student(Id: INTEGER, Name:STRING, Address:STRING, Status:STRING)
Transcript(StudId:INTEGER, CrsCode:STRING, Semester:STRING, Grade:STRING)
```

(a) The following query

```
SELECT S.Name
FROM  STUDENT S
WHERE 3 >
      SELECT  COUNT (*)
      FROM  TRANSCRIPT T
      WHERE S.Id = T.StudId  AND
                  T.Grade = 'A'  AND T.Semester = 'S2002'
```

returns the names of all students who got an A in a more than three course in the Spring semester, 2002. Assuming that the inner and outer queries are optimized separately, what index would you choose for the inner query? What query plan would it be reasonable to expect the DBMS to use?

Solution:

A clustered B^+ tree index on TRANSCRIPT with search key (StudId, Grade, Semester). Scan STUDENT, for each row search the index, the A grades for the corresponding student in S2002 are grouped and can be easily counted.

(b) Convert the query to a non-nested form.

Solution

```
SELECT S.Name
FROM  STUDENT S,  TRANSCRIPT T
WHERE S.Id = T.StudId
GROUP BY S.Id, S.Name, T.Semester, T.Grade
HAVING T.Semester = 'S2002'  AND T.Grade = 'A'
        AND  COUNT(*) > 3
```

(c) Suggest an index for the new query and a possible query plan.

Solution:

One possibility is to use a clustered B^+ tree index with search key Id on STUDENT and the same index as described above on TRANSCRIPT. Since the tables are pre-sorted on the Id attribute, a sort-merge join can be used without having to do any sorting, the A grades can be counted and appropriate student names can be output in a single pass over the tables.

6. The table Professor has attributes Id, Name, Dept, Salary, and Rank and its primary key is Id. Consider the statements

```
SELECT Name, Dept                SELECT Dept, COUNT (*)
FROM Professor                   FROM Professor
WHERE Id = :ssn                  WHERE Salary > sal
                                 GROUP BY Dept
   (S1)                             (S2)
```

where ssn is a host variable and sal is a literal.

(a) Give an efficient query plan for S1 and state what index (search key, clustered, nonclustered, B+ tree, hash) is used.

Solution

Clustered or unclustered, hash or B^+ tree index on Id
Use index to locate unique row satisfying Id = :N

(b) Now suppose, in addition to optimizing S1, we are equally concerned with optimizing the execution of S2. Assuming sal is a large number (most rows do not satisfy the WHERE clause) describe the query plan you would like the optimizer to use and any changes that might need to be made to your answer to the previous part. State what index (search key, clustered, nonclustered, B+ tree, hash) is used for each statement. Explain your reasoning.

Solution

There are two acceptable answers.

Only a few rows satisfy the condition on Salary The query plan for S2 uses an unclustered B^+ tree index on Salary to get the rows satisfying the WHERE clause. Sort them by Dept and count the size of each group. The query plan for S1 given in the previous part is unchanged.

More than a few rows satisfy the condition on Salary The query plan for S2 uses a clustered B^+ tree index on Salary. Scan leaf level starting from sal to get the satisfying rows, sort them by Dept and count the size of each group. The answer to the previous part must now involve an unclustered index on Id.

(c) Redo the previous part assuming that sal is a small number (most rows satisfy the WHERE clause). Explain your reasoning.

Solution

Use a clustered B^+ tree index on Dept. Scan the leaf level, groups will be contiguous and the number of qualifying members of each group can be counted. Since most rows satisfy the WHERE condition most of the pages of Professor will have to be read and sorted. By using a clustered index on Dept the sort is done in advance and fast scanning techniques can be used to scan the table.
The answer to the first part must now involve an unclustered index on Id.

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

1. Translate the following relational algebraic expression into **domain** relational calculus:

$$\pi_{\texttt{Name}}(\textsc{Actor} \bowtie_{\texttt{Id=ActorId}} \sigma_{\texttt{MovieId=666}}(\textsc{Played}))$$

Solution:

$$\{Name \mid \exists Id(\textsc{Actor}(Id, Name) \text{ AND } \textsc{Played}(Id, \text{'666'})) \}$$

2. Consider the following relational schema:

   PUBLISHED(Paper, Journal)
   TOPIC(Journal, Topic)
   AUTHOR(Paper, Researcher)

   Write the following query in **domain** relational calculus: "*Find all researchers who published a paper in every database journal*." (A database journal is one where one of the topics is 'DB'.)

   Solution:

   {r | ∀j Topic(j, 'DB') → ∃p (Published(p, j) AND Author(p, r)) }

3. Consider the following relational schema:

   SUPPLIER(Name, Part)
   PROJECT(Name, Part)

   A tuple, $\langle \mathtt{n}, \mathtt{p} \rangle$, in the first relation means that supplier n has part p. A similar tuple in the second relation means that project n uses part p. Write the following query in **domain** relational calculus: "*Find the names of every supplier who has a part, which is used by every project*."

   Solution:

   $\{\mathtt{s} \mid \exists \mathtt{part}\,(\textsc{Supplier}(\mathtt{s}, \mathtt{part}) \rightarrow \forall \mathtt{pn}\,\textsc{Project}(\mathtt{pn}, \mathtt{part})\}$

4. Consider the following relational schema:

   PLAYED(Actor, Movie)
   DIRECTED(Director, Movie)

   Write the following query in **tuple** relational calculus: "*Find all actors who played in every movie of some director*." (The director can be different in each case.)

   Solution:

   { P.Actor | Played(P) and ∃ D1 ∈ Directed (

   ∀ D2 ∈ Directed (
   D1.Director=D2.Director
   → ∃ P2 ∈ Played(P2.Actor=P.Actor and P2.Movie=D2.Movie)
)
) }

5. Consider the following relation, which describes papers and their authors:

   AUTHOR(Paper, Researcher)

   Two researchers are *direct* collaborators if they have at least one paper that they both co-authored. Two researchers r_1 and r_n are *indirect* collaborators, if there are researchers r_2, ..., r_{n-1} such that r_1 directly collaborated with r_2, r_2 with r_3, ..., and r_{n-1} with r_n.

   Write a recursive SQL query that computes all collaborators of 'John Doe' (direct or indirect).

   Solution:

   First create a view for directly related authors.

```
CREATE VIEW DIRECT(res1, res2) AS
  SELECT A.Researcher, B.Researcher
  FROM AUTHOR A, AUTHOR B
  WHERE A.Paper = B.Paper
        AND A.Researcher <> B.Researcher
```

   Now using above created view create a recursive view for `indirect` collaborators:

```
CREATE RECURSIVE VIEW INDIRECT(res1, res2) AS
  SELECT * FROM DIRECT
  UNION
  SELECT D.res1, I.res2
  FROM DIRECT D, INDIRECT I
  WHERE D.res2 = I.res1
        AND D.res1 <> I.res2
```

   Now you can select all the authors who are related to 'John Doe'.

```
SELECT *
FROM INDIRECT I
WHERE I.res1 = 'John Doe'
```

6. Consider the following relational schema:

   DIRECTFLIGHT(From, To, Distance)

   Write a recursive SQL query that returns tuples of the form ⟨From,To,Distance⟩, which represent *direct or indirect* flights (*i.e.*, flights composed of one or more segments) and the aggregate distance over all segments for each such flight. (If there are several ways to reach B from A and the total distance is different in each case then the output would have several tuples of the form ⟨A,B,...⟩.) The exact query is: "*Find all tuples of the above form provided that the distance is less than 10000.*"

   Solution:

```
WITH RECURSIVE INDIRECTFLIGHT(From,To,Distance) AS
    ( SELECT *
      FROM DIRECTFLIGHT D
      UNION
      SELECT D.From, I.To, D.Distance+I.Distance
      FROM INDIRECTFLIGHT I, DIRECTFLIGHT D
      WHERE D.To = I.From)
SELECT *
FROM INDIRECTFLIGHT I
WHERE I.Distance < 10000
```

7. Consider the following relational schema, which represents immediate components that go into a complex part:

   COMPONENT(Part, DirectSubpart, Quantity)

   Write a recursive SQL query that computes a bill of materials for every part. The bill of materials is a relation of the form

   BILLOFMATERIALS(Part, InDirectAtomicSubpart, Quantity)

   An *atomic* part is one that does not have components of its own.

   In other words, if a complex part is represented as a tree where intermediate nodes are components of the part and children are direct components, then we are interested in the leaves and the number of copies of each leaf needed to build the part.

   Solution:

```
CREATE RECURSIVE VIEW BILLOFMATERIALS(Part,InDirectAtomicSubpart,Quantity)
AS
SELECT *
FROM COMPONENT
UNION
SELECT C.Part, B.InDirectAtomicSubpart, B.Quantity * C.Quantity
FROM COMPONENT C, BILLOFMATERIALS B
WHERE C.DirectSubpart=B.Part
```

Problems for Chapter 14: Object Databases

1. Consider the following schema in SQL:1999.

```
CREATE TABLE Warehouse (
  Address ROW(Street CHAR(20), ZIP CHAR(5)),
  Company CompanyType )


CREATE TYPE CompanyType (
  Name  CHAR(20),
  Phone CHAR(10) )
```

   Insert a tuple into Warehouse, for the company Acme, Inc., which has a phone number 6311237654 and whose warehouse is located on Main St. at ZIP code 44744.

   Solution:

```
INSERT INTO Warehouse
VALUES  (
     ROW ( 'Main St.', 44744 ),
     new CompanyType()
           . Name('Acme Inc.')
           . Phone('6311237654')
         )
```

2. Use SQL:2003 with the MULTISET construct to represent a bank database with UDTs for ACCOUNT and CUSTOMER. The list of attributes should be short, but reasonable for this problem. Formulate the following query:

   Find all customers whose combined balances in all accounts is 500000 or more.

   Solution:

```
CREATE TABLE CUSTOMER (
   Id   CHAR(10),
   Name CHAR(20),
   Accounts Ref( ACCOUNTTYPE) MULTISET SCOPE ACCOUNT
)

CREATE TABLE ACCOUNT OF ACCOUNTTYPE

CREATE TYPE ACCOUNTTYPE (
   Id      INTEGER,
   Balance INTEGER
)

SELECT C.Name
FROM CUSTOMER C
GROUP BY C.Id
HAVING SUM(C->Accounts.Balance) >= 500000
```

3. Use SQL:2003 with the MULTISET construct to represent a database with UDTs for PERSON and EMPLOYEE. Employees are persons and persons have children (use MULTISET to represent the corresponding attribute).

   - Provide the appropriate CREATE TABLE statements.
   - Write the following query: *Find all employees who have at least one child who is also an employee.*

   Solution:

```
CREATE TYPE PERSONT AS (
ID    INTEGER,
Name  CHAR(20),
DOB   DATE,
Children  Ref( PERSONT) MULTISET SCOPE EMPLOYEE,
REF IS ID
)

CREATE TYPE EMPLT UNDER PERSONT AS (
Salary INTEGER
)

CREATE TABLE PERSON OF PERSONT
CREATE TABLE EMPLOYEE OF EMPLT UNDER PERSON

SELECT E.Name
FROM UNNEST(E.Children) C
WHERE C->ID IN ( SELECT E.ID FROM EMPLOYEE E)
```

4. Consider the following ODL definitions:

```
class PERSON : OBJECT ( extent PERSONEXT ) : PERSISTENT;
{
attribute String Name;
... ... ...
}
class STUDENT extends PERSON ( extent STUDENTEXT ) : PERSISTENT;
{
attribute Set< TRANSCRIPTRECORD> Transcript;
... ... ...
}
struct TRANSCRIPTRECORD {
String CrsCode;
float  Grade;
String Semester;
}
```

Write the following query **without** the GROUP BY clause: *"List all students with their corresponding average grade."*

Solution:

```
SELECT S.Name, avg( SELECT T.Grade
                    FROM S.Transcript T)
FROM  STUDENTEXT S
```

5. Consider the following ODL definitions:

```
class ACTOR ( extent ACTOREXT ) : PERSISTENT;
{
  attribute String Name;
  relationship Set< MOVIE> PlayedIn;
  ... ... ...
}
class MOVIE ( extent MOVIEEXT ) : PERSISTENT;
{
   attribute String Title;
   relationship set< ACTOR> Actors;
  ... ... ...
}
```

Write the following query using a nested subquery inside the SELECT clause: *For each actor, A, show the total number of different other actors who played together with A in at least one movie.*

Note that the ODL statements above do not have the `inverse` clause in the relationships `PlayedIn` and `Actors`. Briefly explain how it affects the reliability of the answer to the query that you have composed.

Solution:

```
SELECT DISTINCT Name: A.Name,
         Count: count( SELECT DISTINCT B.Name
                       FROM ACTOREXT B, B.PlayedIn M
                       WHERE M in A.PlayedIn
                ) - 1
FROM ACTOREXT A
```

Without inverse relationship the result of the above query is not reliable. There can be some actor who has played in some movie, but that movie need not be having reference to the actor. Similarly some movie may be there in which some actor played, but there need not be any reference to that movie in actor class. In these cases the result which we compute from above query may not be correct.

6. Consider the following schema

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

   (a) Use the ODL language of the ODMG standard to represent this schema. Adhere to the object-oriented design methodology and use set-valued relationships.

   Solution:

```
class PERSON: OBJECT {
  attribute String Name;
}
class MOVIE: OBJECT
(extent MOVIEEXT)
{
   attribute String Title;
   relationship PERSON Director;
   relationship Set< ACTOR> Stars
          inverse ACTOR::PlaysIn;
}
class ACTOR extends Person
(extent ACTOREXT)
{
   relationship Set< MOVIE> PlaysIn
          inverse MOVIE::Stars;
}
```

   (b) Use the result ontained in (a) to formulate the following query using OQL: *For every actor,* ac*, count the number of movies directed by Joe Schmuck where that actor (*ac*) played a role.*

Solution:

```
SELECT DISTINCT Actor : A.Name
                Count : count( SELECT M
                               FROM MOVIEEXT M
                               WHERE M.Director.Name='Joe Public'
                                     AND A IN M.Stars )
FROM ActorExt A
```

7. Consider the following ODL definitions:

```
class Student ( extent StudentExt ) : PERSISTENT; {
    attribute String Name;
    relationship Set< Course> Enrolled;
    ... ... ...
}
class Course ( extent CourseExt ) : PERSISTENT; {
    attribute String CrsCode;
    relationship set< Student> TAs;
    ... ... ...
}
```

Write the following query **without** the GROUP BY clause: "*List, for each student, the number of different TA names the student has to remember*." (Assume that different TAs can have the same name and each student interacts with every TA in every course the student takes.)

Solution:

```
SELECT struct
              Name: S.Name
              Count: COUNT( SELECT DISTINCT(FLATTEN(E.TAs).Name)
                             FROM S.Enrolled E
                           )

FROM StudentExt S
```

Problems for Chapter 15: XML and Web Data

1. Write a sample XML document, which contains:

   - A list of parts (part name and Id)
   - A list of projects. For each project, a nested list of parts used along with their quantity.

   Write a DTD for this document and an XML schema. Express all key and referential constraints.

   Choose your representation in such a way as to maximize the number of possible key and referential constraints representable using DTDs.

   Solution:

   Document:

```
<?xml version="1.0" encoding="UTF-8"?>
  <document>
    <parts>
        <part id="1" name="screw"/>
        <part id="2" name="bar"/>
        <part id="3" name="bolt"/>
        <part id="4" name="nut"/>
    </parts>
    <projects>
        <project name="assembly">
           <usedPart id="1" qty="15"/>
           <usedPart id="3" qty="20"/>
        </project>
        <project name="construction">
           <usedPart id="2" qty="5"/>
           <usedPart id="3" qty="44"/>
           <usedPart id="4" qty="44"/>
        </project>
    </projects>
  </document>
```

   DTD:

```
<!DOCTYPE document[
  <!ELEMENT document (parts, projects)>
  <!ELEMENT parts (part*)>
  <!ELEMENT projects (project*)>
  <!ELEMENT part EMPTY>
  <!ELEMENT project (usedPart*)>
  <!ELEMENT usedPart EMPTY>
```

```
  <!ATTLIST part
            id ID #REQUIRED
            name CDATA #REQUIRED >
  <!ATTLIST project
            name CDATA #REQUIRED >
  <!ATTLIST usedPart
            id IDREF #REQUIRED
            qty CDATA #IMPLIED >
]>
```

Schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:pr="http://xyz.org/project"
        targetNamespace="http://xyz.org/project">

  <element name="document">
    <complexType>
      <sequence>
        <element name="parts" type="pr:partsType"/>
        <element name="projects" type="pr:projectsType"/>
      </sequence>
    </complexType>
  </element>

  <complexType name="partsType">
    <sequence>
      <element name="part" minOccurs="0" maxOccurs="unbounded">
        <complexType>
          <attribute name="id" type="pr:partIdType"/>
          <attribute name="name" type="string"/>
        </complexType>
      </element>
    </sequence>
  </complexType>

  <complexType name="projectsType">
    <sequence>
      <element name="project" minOccurs="0" maxOccurs="unbounded" >
        <complexType>
          <sequence>
            <element name="usedPart" type="pr:usedPartType"
                     minOccurs="0" maxOccurs="unbounded" />
          </sequence>
          <attribute name="name" type="string" />
        </complexType>
```

```
      </element>
    </sequence>
  </complexType>

  <complexType name="usedPartType">
    <attribute name="id" type="pr:partIdType"/>
    <attribute name="qty" type="integer"/>
  </complexType>
  <simpleType name="partIdType">
    <restriction base="string">
      <pattern value="[0-9]+"/>
    </restriction>
  </simpleType>

  <key name="keyForPart">
    <selector xpath="parts/part"/>
    <field xpath="@id" />
  </key>
  <keyref name="foreignKeyUsedPart" refer="pr:keyForPart">
    <selector xpath="projects/project/usedPart"/>
    <field xpath="@id" />
  </keyref>
</schema>
```

2. Compose a sample XML document, which contains:

   - A list of movie actors (*e.g.*, name, Id).
   - A list of movies. For each movie, indicate the name, year, and a nested list of scenes. Each scene has a name and a list of actors.

   Write a DTD for this document and an XML schema. Use ID, IDREF, and IDREFS (both in the DTDs and the Schema) to express keys and referential constraints.

   Choose your representation in such a way as to maximize the number of possible key and referential constraints representable using the available means.

   Solution:

   Sample XML File:

```
<?xml version="1.0" encoding="UTF-8"?>
    <MovieInfo>
       <Actors>
          <Actor Actor_ID="AP01">
             <FirstName>Joe</FirstName>
             <LastName>Brown</LastName>
          </Actor>
          <Actor Actor_ID="AP02">
             <FirstName>Stan</FirstName>
             <LastName>Stanley</LastName>
          </Actor>
       </Actors>
       <Movies>
          <Movie Name="Movie01" Year="1999">
             <Scenes>
                <Scene Name="sc01">
                   <Participants>
                      <Participant ActorRef="AP02"/>
                   </Participants>
                </Scene>
                <Scene Name="sc02">
                   <Participants>
                      <Participant ActorRef="AP01"/>
                      <Participant ActorRef="AP02"/>
                   </Participants>
                </Scene>
             </Scenes>
          </Movie>
       </Movies>
    </MovieInfo>
```

DTD:

```
    <!DOCTYPE MovieInfo[
        <!ELEMENT MovieInfo (Actors, Movies)>
        <!ELEMENT Actors (Actor+)>
        <!ELEMENT Actor (FirstName, LastName)>
        <!ELEMENT FirstName (#PCDATA)>
        <!ELEMENT LastName (#PCDATA)>
        <!ATTLIST Actor Actor_ID ID #REQUIRED >
        <!ELEMENT Movies (Movie*)>
        <!ELEMENT Movie (Scenes)>
        <!ELEMENT Scenes (Scene+)>
        <!ELEMENT Scene (Participants)>
        <!ELEMENT Participants (Participant+)>
        <!ELEMENT Participant EMPTY>
        <!ATTLIST Participant ActorRef IDREF #REQUIRED >
        <!ATTLIST Movie
           Name ID #REQUIRED
           Year CDATA #REQUIRED
        >
        <!ATTLIST Scene Name CDATA #REQUIRED >
    ]>
```

XML Schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:q4="http://cs.sunysb.edu/cse532/midterm"
        targetNamespace="http://cs.sunysb.edu/cse532/midterm">

 <element name="MovieInfo">
   <complexType>
      <sequence>
         <element name="Actors">
            <complexType>
               <sequence>
                  <element name="Actor" type="q4:Actor_Type"
                                        minOccurs="1" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
         <element name="Movies">
            <complexType>
               <sequence>
                  <element name="Movie" type="q4:Movies_Type"
                                        minOccurs="1" maxOccurs="unbounded"/>
```

```
            </sequence>
          </complexType>
        </element>
      </sequence>
    </complexType>
  </element>

  <complexType name="Actor_Type">
      <sequence>
          <element name="FirstName" type="string" nillable="false"/>
          <element name="LastName" type="string" nillable="false"/>
      </sequence>
      <attribute name="ActorID" type="q4:ActorID_Type" use="required"/>
  </complexType>

  <simpleType name="ActorID_Type">
      <restriction base="ID">
         <pattern value="AP[0-9]+"/>
      </restriction>
  </simpleType>

  <complexType name="Movies_Type">
      <sequence>
         <element name="Scenes" >
            <complexType>
               <sequence>
                  <element name="Scence" type="q4:Scene_Type"
                              minOccurs="1" maxOccurs="unbounded"/>
               </sequence>
            </complexType>
         </element>
      </sequence>
      <attribute name="Name" type="string" use="required"></attribute>
      <attribute name="Year" type="gYear" use="required"></attribute>
   </complexType>

   <complexType name="Scene_Type">
      <sequence>
          <element name="Participant" type="q4:Participant_Type"
                               minOccurs="1" maxOccurs="unbounded"/>
      </sequence>
      <attribute name="Name" type="string"></attribute>
   </complexType>
   <complexType name="Participant_Type">
      <attribute name="ActorRef" type="q4:ActorIDREF_Type" use="required"/>
   </complexType>
```

```
   <simpleType name="ActorIDREF_Type">
      <restriction base="IDREF">
         <pattern value="AP[0-9]+"/>
      </restriction>
   </simpleType>
</schema>
```

3. Compose a sample XML document, which contains the following two kinds of data:

   - *A list of papers* with their titles, dates of publication, journals where they appear, and a list of authors. Do not use child elements to represent all this — only XML attributes are allowed.
   - A list of *authors*, where each author has an Id, name, institution, and a list of papers that the author has published. Authors can have the XML attribute `Id`, but all other information must be represented using XML elements.

   Show a sample document. Write an XML schema for this document. Include the key and foreign key constraints assuming that papers are uniquely identified by their Title and Journal attributes (within the list of papers), while authors are identified using the Id attribute (within the list of authors).

   Solution:

   Sample Document:

```
<publisher>
  <papers>
    <paper title="abc" dateOfPubl="12/03"
           journal="j1" listOfAuths="111 222"/>
    <paper title="def" dateOfPubl="12/02"
           journal="j2" listOfAuths="111 222"/>
    .
    .
  </papers>
  <authors>
    <author id="111">
      <name>
        A1
      </name>
      <institution>
        I1
      </institution>
      <listOfPapers>
        <paper title="abc" journal="j1"/>
        <paper title="def" journal="j2"/>
      </listOfPapers>
    </author>
    <author id="222">
      <name>
        A2
      </name>
      <institution>
        I2
      </institution>
```

```
      <listOfPapers>
        <paper title="abc" journal="j1"/>
        <paper title="def" journal="j2"/>
      </listOfPapers>
    </author>
    .
    .
  </authors>
</publisher>
```

XML Schema:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        xmlns:pbl="http://somepublisher.com/documents"
        targetNameSpace="http://somepublisher.com/documents">
  <element name="publisher">
    <complexType>
      <sequence>
        <element name="papers" type="pbl:papersType"
                 minOccurs="0" maxOccurs="unbounded"/>
        <element name="authors" type="pbl:authorsType"
                 minOccurs="0" maxOccurs="unbounded"/>
      </sequence>
      <key name="PrimaryKeyForPapers">
        <selector xpath="papers/paper"/>
        <field xpath="@title"/>
        <field xpath="@journal"/>
      </key>
      <key name="PrimaryKeyForAuthors">
        <selector xpath="authors/author"/>
        <field xpath="@id"/>
      </key>
      <keyref name="ForeignKeyForPapers" refer="pbl:PrimaryKeyForAuthors">
        <selector xpath="papers/paper"/>
        <field xpath="@listOfAuths"/>
      </keyref>
      <keyref name="ForeignKeyForAuthors" refer="pbl:PrimaryKeyForPapers">
        <selector xpath="authors/author/listOfPapers/paper"/>
        <field xpath="@title"/>
        <field xpath="@journal"/>
      </keyref>
    </complexType>
  </element>
  <complexType name="papersType">
    <sequence>
      <element name="paper" type="pbl:paperType"/>
```

```
    </sequence>
  </complexType>
  <complexType name="authorsType">
    <sequence>
      <element name="author" type="pbl:authorType"/>
    </sequence>
  </complexType>
  <complexType name="paperType">
    <attribute name="title" type="string"/>
    <attribute name="dateOfPubl" type="string"/>
    <attribute name="title" type="string"/>
    <attribute name="listOfAuths" type="IDREFS"/>
  </complexType>
  <complexType name="authorType">
    <attribute name="id" type="ID"/>
    <sequence>
      <element name="name" type="string"/>
      <element name="institution" type="string"/>
      <element name="listOfPapers">
        <complexType>
          <sequence>
            <element name="paper">
              <attribute name="title" type="string"/>
              <attribute name="journal" type="string"/>
            </element>
          </sequence>
        </complexType>
      </element>
    </sequence>
  </complexType>
</schema>
```

4. Write an XSLT stylesheet that traverses the document and preserves all elements, attributes, and text nodes, but discards the `CrsTaken` elements. The stylesheet must not depend on the knowledge of where the `CrsTaken` elements reside in the source document.

   Solution:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xsl:version="1.0" >
    <xsl:template match="CrsTaken">
    </xsl:template>
    <xsl:template match="/|*">
         <xsl:copy>
              <xsl:apply-templates/>
         </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

   The first template discards `CrsTaken` elements and the second copies the root or element nodes and then proceeds recursively. Note that the first template overrides the second when the current node is a `CrsTaken` element. Note that the default template for attributes and text nodes copies them over, so we do not need to specify this action.

5. Consider a document that contains a list of professors (elements with attributes name, Id, department Id) and a separate list of classes taught (elements with attributes professor Id, course code, semester). Use XQuery and aggregate functions to produce a list of professors (name, Id) with the number of courses taught. The same course taught in different semesters counts as one.

   Solution:

```
<result>
  FOR $p = document("...")//professor
  LET $c := { FOR $c = document("...")//classes
              WHERE $c/@ProfId = $p/@Id
              RETURN $c/CrsCode
            }
  RETURN
        <timesTaught name="$p/@Name" id="$p/@Id"
                     numberTaught="count(distinct-values($c))"
</result>
```

6. Write an XSLT stylesheet that traverses the document and preserves all elements, attributes, and text nodes, until it hits an element named `fbar`. Once it enters an `fbar`-element, the stylesheet ignores all attributes and recursively converts elements into text nodes by stripping the element tags. For instance, the document

```
<abc>
   aaa <bbb foo="1"><a>12</a><b>24</b></bbb>
   <fbar>dddd <a b="f">34</a><b><c z="1">ppp</c>kkk</b></fbar>
   fff <qqq foo="1"><aa>12</aa><bb>24</bb></qqq>
</abc>
```

would become

```
<abc>
   aaa <bbb foo="1"><a>12</a><b>24</b></bbb>
   dddd 34 ppp kkk
   fff <qqq foo="1"><aa>12</aa><bb>24</bb></qqq>
</abc>
```

Solution:

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

7. Write an XSLT stylesheet, which traverses documents and does the following:

   If it finds an attribute of some element, *e*, it converts the attribute into a text node, which becomes a child of *e*. The value of the text node is the same as the value of the attribute from which this node was created. The stylesheet preserves all other parts of the input document.

Solution:

```
<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:stylesheet>
```

The second template copies over the root node and element nodes. The default for text and attribute nodes is to copy their values as text, which is precisely what we wanted to do.

8. Write an XSLT stylesheet, which takes a document and produces the same document with all attributes removed.

   Solution:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xsl:version="1.0" >
    <xsl:template match="/|node()">
         <xsl:copy>
              <xsl:apply-templates/>
         </xsl:copy>
    </xsl:template>
</xsl:stylesheet>
```

9. Consider a document of the following form:

```
<project name="foo">
  <supplier name="acme" part="screw"/>
  <supplier name="..." part="..."/>
  ...
</project>
<project ...>
...
</project>
```

Use XQuery to produce a document of the following form:

```
<supplier name="acme" part="screw" projectsSupplied="15">
   <project name="foo"/>
   <project .../>
   ...
</supplier>
<supplier ...>
   ...
</supplier>
```

where projects are grouped according to the suppliers and parts that these projects bought. The attribute `projectsSupplied` specifies the number of projects that bought the given part from the given supplier.

Solution:

```
<result>
  FOR $s = distinct-values(document("...")//supplier)
  LET $proj := { FOR $p = document("...")//project
                 WHERE  $p/supplier/name = $s/name
                        AND  $p/supplier/part = $s/part
                 RETURN
                        <project name=$p/name />
               }
  RETURN
    <supplier name=$s/name, part=$s/part,
              projectsSupplied=count($proj)>
         $proj
    </supplier>
</result>
```

10. Consider a document of the form

```
<book name="..." price="...">
   <store name="..." city="...">
   <store name="..." city="...">
   <store name="..." city="...">
    ...  ...  ...
</book>
<book name="..." price="...">
   <store name="..." city="...">
   <store name="..." city="...">
   <store name="..." city="...">
    ...  ...  ...
</book>
...  ...  ...
```

Write the following query using XQuery:

Find the city with the **highest** *average price of books.*

Solution:

```
DECLARE FUNCTION AvgPrice() AS element()* {
   FOR $c in distinct-values(document(...)/book/store/@city)
   RETURN
      <cityAvg city=$c avg=avg(
             FOR $b in document(...)/book
             WHERE SOME $b/store/@city=$c
             RETURN $b/@price
                              ) />
}

FOR $c in AvgPrice()
WHERE  $c/@avg = max(FOR $cc in AvgPrice() RETURN $cc/@avg)
```

SOME is used in the above so that any book will be returned only once if it happens to be sold in several stores in the same city.

Problems for Chapter 16: Distributed Databases

1. Suppose that we have two relations:

   ORDER(Customer,ItemNumber,Quantity)
   INVENTORY(ItemNumber,ItemName,Price)

   The ORDER relation does not have non-trivial keys. In INVENTORY, ItemNumber is a key, but ItemName is not. The database is distributed so that ORDER is at site A and INVENTORY is at site B. The answer to the query

   $$\pi_{\texttt{Customer,Price}}(\sigma_{\texttt{ItemName='screwdriver'}}(\text{ORDER} \bowtie \text{INVENTORY}))$$

   is to be sent to site C.

   Assume the following statistics:

   - Every attribute is 20 bytes long.
   - ORDER has 10,000 tuples; 500 different item numbers.
   - INVENTORY has 1,000 tuples; 200 different item names.

   Find the best evaluation plan for this distributed query and estimate its cost in terms of bytes that need to be sent around.

   Solution:

   $\sigma_{\texttt{ItemName='screwdriver'}}$(INVENTORY) has 1000/200 = 5 tuples on the average.

   We send the projection on ItemNumber,Price of this selection to site A. The cost is 5*(20+20)=200 bytes.

   Join at site A: The average number of tuples in ORDER that matches a given ItemNumber is 10000/500=20. Since we have sent at most 5 item numbers to site A, the join at that site will have at most 5*20=100 tuples.

   Send Price and Customer of each of those 100 tuples to site C. Cost: 100*(20+20)=4000.

   Total cost: 200+4000=4200.

2. Rewrite the following queries using the semijoin operator (and possibly other operators):

   (a) $\pi_{\texttt{Name}}(\textsc{Student} \bowtie_{\texttt{Id=StudId}} \textsc{Transcript})$

   (b) $\pi_{\texttt{Name,Grade}}(\textsc{Student} \bowtie_{\texttt{Id=StudId}} \textsc{Transcript})$

   Assume that STUDENT has the attributes `Id` and `Name`, and TRANSCRIPT has `Id`, `CrsCode`, `Semester`, and `Grade`.

   Solution:

   (a) $\pi_{\texttt{Name}}(\texttt{Student} \ltimes_{\texttt{Id=StudId}} \texttt{Transcript})$

   (b) $\pi_{\texttt{Name,Grade}}((\texttt{Student} \ltimes_{\texttt{Id=StudId}} \texttt{Transcript}) \bowtie \texttt{Transcript})$

Problems for Chapter 17: OLAP and Data Mining

1. Suppose that we have 3 dimension tables each having 20 tuples.

   (a) What is the maximum number of rows in the fact table of this application?

   (b) What is the maximal number of rows in the join index for joining one of these dimension tables with the fact table?

   Explain your answers.

   Solution:

   (a) Each tuple of points, which contains one point from each dimension defines a point in the hypercube. Therefore, the maximal number of facts is the "volume" of the hypercube, *i.e.*,

   $$\begin{aligned} &\#\text{ of tuples in dim 1} \;\times\; \#\text{ of tuples in dim 2} \;\times\; \#\text{ of tuples in dim 3} \\ &= \; 20 \times 20 \times 20 \; = \; 8000 \end{aligned}$$

   (b) Since the `Id` attribute of the dimension table is a key in that table, it follows that the number of tuples in the join of that dimension table with the fact table equals the number of tuples in the fact table. Since a join index has one tuple per tuple in the join, the number of tuples in that index must be 8000.

2. Consider the table

TransactionId	Product
1	pen
1	ruler
1	bread
2	ruler
2	bread
3	chicken
3	yogurt
3	pen
4	pen
4	ruler
4	paper
5	bread
5	ruler
6	bread

Perform a priori algorithm to find two-item associations with support of at least 30%. Determine the confidence of each association that you found.

Solution:

Co-occurrences of pairs of items:

pen, ruler – 2
pen, bread – 1
ruler, bread –3

Since there are 6 transactions, the support for pen-ruler is 33% and for ruler-bread is 50%.

Occurrences of single items:

pen occurs in 3 transactions
ruler in 4
bread in 4

Therefore, support is as follows:

pen → ruler : 2/3 = 75%
ruler → pen : 2/4 = 50%
ruler → bread : 3/4 = 75%
bread → ruler : 3/3 = 75%

Problems for Chapter 18: ACID Properties of Transactions

1. Short answer questions.

   (a) What properties must a consistent transaction satisfy?

   Solution:

   Transform database from one consistent state to another.
   The final state must satisfy the transaction's specifications with respect to the initial state.

   (b) What property must an atomic transaction satisfy?

   Solution:

   Either the transaction must run to completion or it must have no effect at all.

2. Give an example of a schedule of two transactions, T_1 and T_2, whose database operations are interleaved, but the execution is isolated in that the overall effect of the two transactions is the same as if the transactions had executed serially with T_1 executing before T_2.

   Solution:

T_1:		$r_1(x)$		$w_1(y)$
T_2:	$r_2(x)$		$w_2(x)$	

   Since T_1 sees the value of x before T_2 wrote it and since T_2 does not access y, the effect is the same as if T_1 had executed to completion before T_2 started.

3. Give an example of a transaction that maintains all the database integrity constraints but nevertheless is not correct in that it does not satisfy its requirements

   Solution:

   A transaction to deposit money in a bank account. One of the arguments is the amount to be deposited and one of its requirements is that it increment the balance field of the account table for that customer by the amount to be deposited. Instead the transaction only increments the balance field by one half of the amount in its argument. (Or maybe it deposits the amount in a different customers account.)

4. What do people mean when they say a transaction is a “unit of work”?

   Solution:

   It updates the database in a way to preserve all integrity constraints and maintains correctness. In other words it is consistent.

5. Give examples of situations in which an ordinary operating system does not guarantee:

   (a) Atomicity

   Solution:

   A program write information into two files, and the system crashes after the first write and before the second write. When the system recovers, the first write will still be there.

   (b) Durability

   Solution:

   A program writes into a file and completes. Then the system crashes before the next system backup (which is done only once a day). The new information in the file is lost.

Problems for Chapter 19: Models of Transactions

1. Short answer questions:

   (a) Consider the following nested transaction:

   $begin\_transaction_1$;
   $statement_{11}$;
   $begin\_transaction_2$;
   $statement_2$;
   $commit_2$ or $abort_2$;
   $statement_{12}$;
   $commit_1$ or $abort_1$;

   Suppose the database system supports savepoints, but not nesting. What statements can be used to replace $begin\_transaction_2$, $commit_2$ and $abort_2$ to get the effect of a nested transaction?.

   Solution

   For $begin\_transaction_2$ use $x := create\_savepoint()$
   For $commit_2$ use an empty statement
   For $abort_2$ use $rollback(x)$

   (b) Under certain circumstances a first-come-first-served recoverable queue can cause entries to be handled in a non-first-come-first-served manner. Explain how this can happen.

   Solution

   T_1 $enq(e_1)$ *commit*
   T_2 $enq(e_2)$ *commit*
   T_3 $e_1 := deq()$ *abort*
   T_4 $e_2 := deq()$ *commit*

   T_3 deletes the entry e_1, at the head of the queue but later aborts and e_1 is replaced. In the intervening time the next entry, e_2 is deleted and serviced.

   (c) In the locking implementation of a recoverable queue both long and short term locks are used, and an entry that has been enqueued is locked differently from the head/tail pointers. Explain.

   Solution

   head/tail pointer uses short term lock to enable concurrent access to queue; enqueued entry uses long term lock since isolation requires that it not be deleted until transaction commits.

   (d) Commit is conditional in the nested transaction model. Explain what this means.

   Solution

   A subtransaction may commit, but if its parent aborts the subtransaction will also be aborted.

2. The model of nested transactions implemented by the Sybase DBMS is different than the one we discussed in the text (which is the standard model). In the Sybase model

   - The children of a parent transaction or subtransaction cannot execute concurrently (the standard model allows children to executed either sequentially or concurrently).
   - When a subtransaction commits, the commit is conditional on the commit of its parent (and ultimately on the commit of the top-level transaction)–as in the standard model
   - When a subtranaction aborts, the entire transaction aborts, which is different than the standard model

   Sybase also implements savepoints (with the same semantics as we discussed in the text). Describe how the standard model of nested transactions (with the limitation that children cannot execute concurrently) can be implemented in Sybase using the Sybase model of nested transactions together with the Sybase implementation of savepoints.

   Solution:

   When a subtransaction starts, it declares a savepoint before its first statement. Later if it wants to abort, it first rolls back to its initial savepoint and then (conditionally) commits.

3. Suppose a transaction, T, uses declarative demarcation and calls two modules, M_1 and M_2, each specified with *RequiresNew*.

   (a) Give an example of a situation in which the overall transaction, T, including the modules is not isolated.

   Solution:

   Suppose T calls M_1, which accesses some data and then commits and gives up its locks. Later T calls M_2, which access the same data that M_1 had accessed and then commits. Between the two accesses some other transaction, T', had accessed and changed that data. M_1 and M_2 have seen different versions of the same data, and hence T (including the two modules) is not isolated.

   (b) Explain why this situation cannot occur if T is a nested transaction.

   Solution: When the first subtransaction (conditionally) commits, it returns its locks to its parent, which later gives them to the second subtransaction. Therefore no other transaction could have accessed the data between the execution of the two subtransactions.

   Note: This answer requires knowledge of the locking implementation of nested transactions given in Chapter 20.

Problems for Chapter 20: Implementing Isolation

1. Short answer questions

   (a) Consider the following three schedules. If a schedule is conflict equivalent to a serial schedule, give the equivalent serial order. If a schedule is view, but not conflict equivalent to a serial schedule give the equivalent serial schedule and explain why it is not conflict serializable. If a schedule is neither conflict nor view equivalent to a serial schedule explain why.

   (1) $r_1(x)\ w_2(y)\ r_2(x)\ r_3(y)\ w_1(z)\ r_1(y)\ w_3(z)$

   (2) $r_2(x)\ r_1(y)\ w_2(y)\ r_3(z)\ w_3(x)\ r_1(z)\ r_1(x)$

   (3) $r_1(y)\ w_2(x)\ w_2(y)\ r_3(z)\ w_1(x)\ r_2(z)\ w_3(x)$

   Solution

   (1) conflict equivalent to $T_2\ T_1\ T_3$
   (2) not serializable since $T_2 \rightarrow T_3$, $T_3 \rightarrow T_1$, $T_1 \rightarrow T_2$ and all conflicts are read/write conflicts
   (3) view equivalent to $T_1\ T_2\ T_3$. Although there is a cycle in the serialization graph, the conflict between $w_2(x)$ and $w_1(x)$ does not affect the final value of x since x is updated by T_3 later.

2. (a) Given a schedule, S, explain how a serialization graph is constructed.

Solution

Each node represents a transaction in S. There exists an edge from node T_1 to T_2 if T_1 and T_2 have conflicting operations in S and T_1's operation precedes T_2's operation

(b) Prove that if the serialization graph has a loop, then S is not serializable.

Solution

Suppose the graph has a loop: $T_1 \rightarrow T_2 \rightarrow \cdots \rightarrow T_n \rightarrow T_1$. Because of the conflicts implied by the graph T_1 must precede T_2 in any equivalent serial order, T_2 must precede $T_3, \cdots, T_n$ must precede T_1. This is a contradiction.

(c) Draw a serialization graph for the schedule S

$$w_1(x)\ r_1(x)\ w_3(z)\ r_2(x)\ r_2(y)\ r_4(z)\ r_3(x)\ w_3(y)$$

If S is serializable give an equivalent serial schedule, if not explain why.

Solution

nodes: $T_1,\ T_2,\ T_3,\ T_4$
directed edges: $(T_1, T_2),\ T_2, T_3),\ (T_1, T_3),\ (T_1, T_4)$
S is serializable with equivalent serial schedule $T_1,\ T_2,\ T_3,\ T_4$.

(d) Consider the following schedule obtained from schedule S of the previous part by adding commit operations.

$$w_1(x)\ r_1(x)\ w_3(z)\ c_1\ r_2(x)\ r_2(y)\ r_4(z)\ c_4\ c_2\ r_3(x)\ w_3(y)\ c_3$$

Is the schedule strict? Explain.

Solution

not strict - $r_4(z)$ is a dirty read

(e) Consider a concurrency control that implements two-phase locking and does not release the write locks that a transaction has acquired until the transaction commits. Does the concurrency control produce only serializable schedules (yes or no)? Does the control produce only strict schedules (explain your answer)?

Solution

serializable - yes
strict - yes because write locks are not released until commit time. Thus a transaction can't read or write an itme that has been written be an uncommitted transaction.

3. Consider the schedule S:

$$r_1(x)\ w_1(z)\ r_2(y)\ r_1(y)\ c_1\ r_3(z)\ w_2(x)\ r_3(y)\ w_2(y)\ c_3\ c_2$$

where *c* represents the commit of a transaction.

(a) Draw a serialization graph for S (you should ignore the commits for this).

Solution

Directed edge from: T_1 to T_2 and T_3, and from T_3 to T_2

(b) Is S serializable? If so, what is the equivalent serial order? If not, explain why.

Solution

serializable in the order T_1, T_3, T_2

(c) Could S be produced by a locking concurrency control using automatic locking? Explain.

Solution

No - T_2 writes *y* before T_3 commits and releases its read lock.

(d) Could S be produced by a concurrency control that implements two-phase locking? Explain.

Solution

Yes - T_3 can release its read lock on *y* after reading *y* because it acquires no further locks.

(e) Suppose c_1 and $r_3(z)$ were interchanged. Is S serializable? Is it strict? Explain.

Solution

S is still serializable since the conflicts and their order is unchanged.
S is not strict since $r_3(z)$ is dirty.

4. (a) A registration system maintains, for each class, a list of the Ids of registered students and a registration count (the length of the list). There is no limit on class size and only two operations are provided:

- *Reg*(*ident*, *class*) - registers the student with Id *ident* in class *class* (if the student is already on the list the operation has no effect)
- *Dereg*(*ident*, *class*) - deregisters the student with Id *ident* in class *class* (if the student is not on the list the operation has no effect)

Indicate with x the conflicts between two operations on the same class in the following conflict table. Explain each conflict that you have shown.

	Reg	*Dereg*
Reg		(x)
Dereg	(x)	

Solution

The requested conflicts are shown in the table above. If *Reg* and *Dereg* have the same arguments (both *ident* and *class*) they don't commute: If *Dereg* comes first the final state has *ident* on the list, if *Reg* comes first it does not

(b) How might a deadlock occur in the system described in the previous part?

Solution

One possibility is T_1 gets a *Reg* lock on $class_1$ and T_2 gets a *Dereg* lock on $class_2$. Then T_1 requests a *Reg* lock on $class_2$. Then T_2 requests a *Dereg* lock on $class_1$

(c) Suppose the system in the first part of the problem is modified by adding the operation *RemoveLast*() to reduce class size by deregistering the last student who has registered. Indicate with x the conflicts between two operations on the same class in a conflict table. Explain each conflict that you have shown.

Solution

	Reg	*Dereg*	*RemoveLast*
Reg	x	x	x
Dereg	x		x
RemoveLast	x	x	

- Two *Reg*'s conflict since the order of Ids on the list is significant (since *RemoveLast* operates on the end of the list)
- *Reg* and *Dereg* conflict since the order of execution makes a difference if both have the same Id
- *Reg* and *RemoveLast* conflict since, depending on the order of execution, the argument of *Reg* will or will not be deleted
- *Dereg* and *RemoveLast* conflict since the order of execution makes a difference if *Dereg* refers to the end of the list

5. The transaction schedules for transactions T_1 and T_2 are:

T_1: $a_1(y)$ $b_1(x)$
T_2: $a_2(x)$ $a_2(y)$ $c_2(x)$

where a, b, and c are abstract operations with conflict table:

requested mode	granted mode a	b	c
a		X	
b	X		
c			X

Consider the following schedule, S, produced by the interleaved execution of T_1 and T_2:

S: $a_1(y)$ $a_2(x)$ $a_2(y)$ $b_1(x)$ $c_2(x)$

(a) Draw a serialization graph for S.

Solution

single edge from T_2 to T_1

(b) What is the condition on a serialization graph that guarantees that a schedule is serializable? Is S serializable?

Solution

no cycles. S is serializable

(c) Would S be allowed by a pessimistic concurrency control that guarantees serializability? Explain.

Solution

S would not be allowed. The control follows the rule that it does not grant a request that creates an ordering among active transactions. Hence, T_1's request for $b_1(x)$ would be made to wait.

(d) How would transaction abort be handled in a schedule involving a, b, and c?

Solution

Each abstract operation has to have an inverse. If a transaction aborts, the inverses of all operations that it has executed are executed in reverse order.

(e) Suppose T_1 and T_2 are multilevel transactions. The transactions (the application level programs that invoke a, b, and c) run at level 2. The programs that implement a. b, and c run at level 1 and they, in turn invoke lower level operations that are implemented at level 0. Level 1 and level 0 each have their own concurrency control. Which concurrency control uses the above conflict table? Why is the concurrency control at level 0 needed?

Explain.

Solution

The above conflict table is used at level 1.
The control at level 0 is needed to ensure that the execution of a program at level 1 that implements a, b, or c is isolated with respect to the execution of concurrent programs at level 1.

(f) Suppose T_1 and T_2 are nested transactions. a, b, and c are now procedures that are invoked by T_1 and T_2 as subtransactions. These subtransactions, in turn, invoke sub-subtransactions. Are there still two concurrency controls? Do the concurrency control(s) use the above conflict table? Explain.

Solution

There is a single concurrency control.l It does not use the conflict table. A conventional concurrency control using shared and exclusive locks is used for all access to the database.

6. (On optional material) The decrement operation, $Dec(x)$, decrements the integer x by 1. It returns no result and always succeeds. The test-and-increment operation, $TestInc(x)$, compares the value of x to a fixed value, N, increments x if $x < N$ and returns OK, or doesn't increment it if $x \geq N$ and returns NO. We will assume that x is initialized to a value less than or equal to N.

   (a) Do two $TestInc$ operations on the same integer, x, commute? Explain.

   Solution No. If $x = N - 1$ interchanging two $TestInc$ operations will result in reversing the response (OK/NO) returned to the two transactions.

   (b) Suppose you built a concurrency control in which $TestInc(x)$ was treated as two partial operations, $TestIncOK(x)$ (which is the operation $TestInc(x)$ when $x < N$) and $TestIncNO(x)$ (which is the operation $TestInc(x)$ when $x \geq N$). Fill in the following conflict table.

	TestIncOK	*TestIncNO*	*Dec*
TestIncOK			
TestIncNO			
Dec			

   Solution

	TestIncOK	*TestIncNO*	*Dec*
TestIncOK		?	X
TestIncNO	X		?
Dec		X	

   The question mark indicates that the requested operation is undefined. Student should either say this or indicate a conflict.

7. Give examples of schedules of read and write operations involving two transactions that are

   (a) Serializable but not strict

   Solution

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

   (b) Strict but not serializable

   Solution

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

   (c) Serializable but not recoverable (include commit and/or abort operations)

   Solution:

   $w_2(x)\ r_1(x)\ w_1(y)\ commit_1\ w_2(z)\ abort_2\ or\ commit_2$

   (d) Recoverable but not serializable (include commit and/or abort operations)

   Solution:

   $r_2(x)\ w_1(x)\ r_1(y)\ commit_1\ w_2(y)\ abort_2\ or\ commit_2$

   (e) Neither serializable nor recoverable (include commit and/or abort operations)

   Solution:

   $w_2(x)\ r_1(x)\ r_1(y)\ commit_1\ w_2(y)\ abort_2\ or\ commit_2$

   (f) Serializable, but not strict

   Solution:

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

8. Give examples of schedules in which

   (a) A pessimistic concurrency control makes a transaction wait, but later allows it to commit, while an optimistic

   concurrency control restarts the transaction.

   Solution:

   $r_2(y)\ w_1(x)\ r_2(x)\ commit_1\ request\_to\_commit_2$

   (b) A pessimistic concurrency control makes a transaction wait, but then allows it to commit, while an optimistic

   concurrency control allows the transaction to commit without waiting.

   Solution:

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

   (c) A two-phase locking concurrency control accepts that schedule, but a timestamp-ordered concurrency control does not

   Solution:

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

Problems for Chapter 21: Isolation in Relational Databases

1. Short answer questions:

   (a) Two transactions running at SERIALIZABLE access a single table. Describe two ways in which they might deadlock with one another.

   Solution

   i. Both get read locks on table, then both try to get write locks

   ii. Table uses row locking. Both write to individual rows and reach the lock escalation threshold and attempt to trade the locks they hold for a table lock.

   (b) What is the weakest isolation level (among the SQL standard levels) that enforces strictness? Assuming a locking implementation, how do the locks at that level do it?

   Solution

   READ COMMITTED - long term write locks prevent dirty writes, short term read locks prevent dirty reads

   (c) Suppose a consistent transaction is decomposed into a chained sequence of smaller transactions. What is the advantage of doing this? What problems are introduced with respect to atomicity, consistency, and isolation?

   Solution

   Advantage: not all updates are lost in a crash, early lock release

   Problems:

   atomicity only part of the work is committed if a crash occurs,

   consistency database might be left in an inconsistent state if a crash occurs

   isolation one transaction might see the intermediate state produced by another (even if no crash occurs)

   (d) Show, by example, that you know what the lost update and write skew anomalies are.

   Solution

   lost update: $r_1(x)\ r_2(x)\ w_1(x)\ w_2(x)$

   write skew: $r_1(x)\ r_1(y)\ r_2(x)\ r_2(y)\ w_1(x)\ w_2(y)$

2. (a) What conditions must be met in order for two operations of different transactions to commute? Be specific.

Solution

When executed in either order and for any starting state they must return the same values to each transaction and they must leave the database in the same final state

(b) A deposit operation, *int dep*(*acctnum*, *amt*) deposits *amt* dollars into the account with number *acctnum* and returns the new balance. A check credit operation

boolean check_cr(*acctnum*, *minbal*)

checks whether the balance in the account with number *acctnum* exceeds *minbal* dollars and returns true if so and false if not.

i. Do two deposit operations on the same account commute? (ans - no)

ii. Do two check credit operations on the same account commute? (ans - yes)

iii. Do a deposit and a check credit operation on the same account commute? (ans - no)

(c) Assume Table1 has two integer attributes, A1 and B1, and Table2 has three integer attributes, A2, B2, and C2. Assume that the domains of A1 and A2 and of B1 and B2 are the same. Consider the statement St:

```
INSERT INTO Table1 (A1, B1)
    SELECT T2.A2, T2.B2
    FROM Table2 T2
    WHERE T2.C2 > 50
```

i. A phantom can arise between a statement St1 that accesses Table1 and St. Give an example of St1.

Solution

```
SELECT ..any attributes of Table1..
FROM Table1
WHERE ..any condition..
```

ii. A phantom can arise between a statement St2 that accesses Table2 and St. Give an example of St2.

Solution

```
INSERT INTO Table2 (A2, B2, C2)
VALUES (?, ?, n)
```

where n is greater than 50

3. You are given the following transactions:

T1: begin transaction r(x, X); if X = 5 then X:= X+1; w(x, X); commit
T2: begin transaction r(x, X); if X = 5 then Y := 17; w(y, Y); commit
T3: begin transaction r(x, X); if X = 5 then X := X-1; w(x, X); commit
T4: begin transaction r(x, X); if X ¡ 5 then X := X-1; w(x, X); commit
T5: begin transaction r(x, X); if X = 5 then
{X := X+1; Y := 17; w(x, X); w(y, Y)}; commit

In each of the following parts two of these transactions are specified to run concurrently at the same isolation level - either READ COMMITTED or REPEATABLE READ. In each of these cases you are to say (for any possible initial values of database items x and y) whether a lost update or deadlock can occur, whether all schedules are serializable. For example, if at a particular isolation level and with a particular initial state of x and y, a non-serializable schedule is possible, answer yes.

	Deadlock	Lost update	Non-serializable
(a) T1 and T2			
READ COMMITTED	ans=no	ans=no	ans=no
REPEATABLE READ	ans=no	ans=no	ans=no
(b) T1 and T3			
READ COMMITTED	ans=no	ans=yes	ans=yes
REPEATABLE READ	ans=yes	ans=no	ans=no
(c) T1 and T4			
READ COMMITTED	ans=no	ans=no	ans=no
REPEATABLE READ	ans=no	ans=no	ans=no

(d) Consider running T1 and T5 at READ COMMITTED. Is deadlock possible? Can an update be lost? Are all schedules serializable? Can the final state produced by any schedule be other than a state produced by a serializable schedule? Justify your answers with an explanation or an example.

Solution

no deadlock since read locks are short term
no lost update
not serializable - consider the schedule $r_1(x)\ r_5(x)\ w_5(x)\ w_5(y)\ w_1(x)$
All final states, however, can always be reached by some serial schedule. If in the intial state $x \neq 5$ all schedules of the two transactions do not change the values of x and y. If in the initial state $x = 5$ then with serial execution the final state is $x = 6$ or $x = 17$. With a non-serializable schedule the final state is $x = 17$.

4. An airlines database has two tables:

   FLIGHTS with attributes flt_num, plane_id, num_reserv
   PLANES with attributes plane_id, num_seats

   The attributes have the obvious meaning. A reservation transaction contains the following steps:

```
            SELECT F.plane_id, F.num_reserv
            INTO :p, :n
            FROM FLIGHTS F
            WHERE F.flt_num = :f
A
            SELECT P.num_seats
            INTO :s
            FROM PLANES P
            WHERE P.plane_id = :p
B
            check that n < s
C
            UPDATE FLIGHTS F
            SET F.num_reserv = :n + 1
            WHERE F.flt_num = :f
D
            commit
```

   Assume that each individual SQL statement is executed in isolation, that the DBMS uses intentions locking and sets locks on tables and rows, and that host variable f contains the number of the flight to be booked. The transaction should not overbook the flight.

   (a) Assuming the transaction is run at READ COMMITTED, what locks are held at points A, B, and D?

   Solution

   A - no locks
   B - no locks
   D - X lock on tuple in FLIGHTS, IX lock on FLIGHTS

   (b) The database can be left in an incorrect state if concurrently executing reservation transactions running at READ COMMITTED are interleaved in such a way that one transaction is completely executed at point B in the execution of another. Describe the problem.

   Solution

   lost update - Both transactions read the same value of num_reserv, both increment their copies and both write the incremented initial value back.

(c) In an attempt to avoid the incorrect execution described in (b) the SET clause of the UPDATE statement is changed to F.num_reserv = F.num_reserv + 1. Can reservation transactions now be run correctly at READ COMMITTED? Explain.

Solution

Flight can now be overbooked. Both transactions read the same value of num_reserv and decide there exists one available seat. Both increment the value in the database (so that the correct count is recorded). But if the initial value indicated that only one seat was left overbooking will result.

(d) Assuming the transaction is run at REPEATABLE READ, what locks are held at points A, B, and D?

Solution

A - S lock on tuple in FLIGHTS, IS lock on FLIGHTS
B - locks held at A plus S lock on tuple in PLANES and IS lock on PLANES
D - locks held at B plus X lock on tuple in FLIGHTS and IX lock on FLIGHTS

(e) What problem does the interleaving of (b) cause at REPEATABLE READ? Explain.

Solution

deadlock: both transaction get S lock on tuple in FLIGHTS and later try to get X lock on the tuple

(f) Would the interleaving of (b) cause an incorrect state if the transaction (either version) were run using SNAPSHOT ISOLATION? Explain.

Solution

No - Each gets a snapshot. The second to request commit is aborted since the first has written to the tuple in FLIGHTS between the time the second read it and the time it requested to commit.

(g) In order to keep track of each passenger a new table, PASSENGER, is introduced that has a row describing each passenger on each flight with attributes name, flt_num, seat_id. SQL statements are appended to the end of the transaction to (1) read the seat_id's assigned to each passenger on the flight specified in f and (2) insert a row for the new passenger that assigns an empty seat to that passenger. What is the weakest (ANSI) isolation level at which the transaction can be run without producing an incorrect state (i.e., two passengers in the same seat)? Explain.

Solution

REPEATABLE READ - a phantom is not possible since only one transaction can add a tuple for a particular flight at a time (a transaction must hold an X lock on the flight tuple before in can insert a row in PASSENGER)

Transactions will run correctly at SERIALIZABLE, but this is not the weakest isolation level.

5. Consider the following statements that access a table, Table, with integer attributes, A and B.

SELECT COUNT (*)	SELECT COUNT (*)	UPDATE Table
FROM Table	FROM Table	SET A = A + 2
WHERE 0 <= B AND B <= 10	WHERE 5 <= B AND B <= 15	
(S1)	(S2)	(S3)

(a) Transaction T1 has statements S1 followed by S2 embedded in it, and transaction T2 has S3 embedded in it and both transactions are consistent. Assume the statements are scheduled by a concurrency control that uses granular locking on rows and tables. In each of the following, give a one or two sentence explanation of your answer.

i. Can S3 be interleaved between S1 and S2 when T1 and T2 are run at REPEATABLE READ?
Solution
No. S1 acquires a long term shared lock on some rows and an IS lock on Table. S3 needs a long term exclusive lock on Table that conflicts with the IS lock

ii. Can S3 be interleaved between S1 and S2 when T1 and T2 are run at READ COMMITTED?
Solution
Yes. T1 releases all locks when S1 completes allowing T2 to execute S3

iii. The specification of T1 asserts that its only function is to return the number of rows satisfying the condition $0 \leq B \leq 10$ and the number of rows satisfying the condition $5 \leq B \leq 15$. The specification of T2 asserts that its only function is to increment A in all rows by 2. Will T1 and T2 satisfy their specifications when both are run at REPEATABLE READ? At READ COMMITTED?
Solution
REPEATABLE READ: yes since no interleaving is allowed
READ COMMITTED: yes since S3 commutes with both S1 and S2 and therefore any interleaved schedule is equivalent to a serial schedule

(b) Now consider the following statement.
(S4) INSERT INTO Table (A, B) VALUES (:x, :y)
where x and y are host variables. Transaction T3 has the single statement S4 embedded in it.

i. What is the maximum range of x and y that guarantees that each schedule of T1 and T3 is equivalent to a serial schedule (serializable)? Explain.
Solution
No restriction on x. Restrictions on y are $y < 5$ or $y > 10$. If $y < 5$, S4 commutes with S2. If $y > 10$, S4 commutes with S1.

ii. Suppose predicate locking is used to implement the isolation level SERIALIZABLE. Will any serializable schedules be disallowed? Explain.
Solution
Yes. The schedule S1, S4, S2 is disallowed if $y = 1$ since S1 gets a long term predicate lock on the predicate $0 \leq B \leq 10$ and S4 attempts to insert a row satisfying the predicate. However, S4 commutes with S2.

iii. Assume an index on B and assume that the concurrency control uses index locking to prevent phantoms. What locks are acquired by S1 and S4? Explain how phantoms are prevented.

Solution

T1 gets an IS lock on Table, S locks on satisfying rows, and S locks on index leaf pages containing entries satisfying $0 \leq B \leq 10$. T2 requires an IX lock on Table, an X lock on the inserted row, and an X lock on the leaf page into which the index entry for the new row will be inserted. The X lock on the index leaf page is not granted if $0 \leq y \leq 10$ since that page will have been locked by T1.

6. A table, Table, is created using

```
CREATE TABLE Table(
    attr1 INT,
    attr2 CHAR (10) )
```

Transaction T executes the statement

```
SELECT T.attr1
    FROM Table T
    WHERE attr2 = 'Mary'
```

(a) What statement does a concurrent transaction have to execute to cause a phantom with respect to this statement?

Solution

INSERT INTO TABLE VALUES (?, 'Mary')

(b) Assuming the table has no indices, what locking strategy can a DBMS use to prevent phantoms?

Solution

lock entire table

(c) If a table has indices, then instead of using the strategy of (b), phantoms can often be prevented using index locking.

i. If Table has an index on attr1, what index pages would have to be locked to prevent a phantom? Explain.

Solution

All index pages since a tuple in which attr2 = 'Mary' could be referenced from any index page. (Alternate answer: There is no good way to lock this index; lock the entire table instead.)

ii. If Table has an index on attr2 what index pages would have to be locked to prevent a phantom? Explain.

Solution

Only the page(s) having a reference to a tuple in which attr2 = 'Mary' might be stored.

(d) Assume two tables, Table1 and Table2, are created, each having three integer attributes and consider the statement

```
SELECT T1.attr1, T2.attr1
    FROM Table1 T1, Table2 T2
    WHERE T1.attr2 = T2.attr2 AND T1.attr3 = 5 AND T2.attr3 = 7
```

Give a single insert statement that might cause a phantom.

Solution

One possibility is
INSERT INTO T1 (?, ?, 5)

7. (a) An application involves a transaction, Trans, containing the statement

```
SELECT T.attrib INTO :y
FROM Table T
WHERE T.key = '123456789'
```

and, at a later point, the statement

```
UPDATE Table
SET attrib = :x
WHERE key = '123456789'
```

where the attribute "key" is the primary key of Table. You can assume that the value in the host variable x is calculated within Trans and related to the value returned by the first statement in host variable y. Instances of Trans are run concurrently with each other. If Trans is executed at READ COMMITTED lost updates are possible. Assuming the standard locking implementation of isolation levels, explain why by describing how locks are acquired and released. Are lost updates possible if Trans is executed at REPEATABLE READ? Explain.

Solution

Short term read locks are used at READ COMMITTED. An instance of Trans releases the lock on the row designated by key when the SELECT completes. A second instance may read the same value of attrib before the first executes the UPDATE statement and both can then update attrib. This sequence is not possible at REPEATABLE READ since read locks are not released until commit.

(b) Suppose that the update statement of the previous part is modified to:

```
UPDATE Table
SET attrib = :x
WHERE T.key = '123456789' AND attrib = :y
```

The new UPDATE can be used instead of the previous UPDATE as one part of a modification of Trans whose purpose is to eliminate lost updates at READ COMMITTED. Explain the complete modification and why it eliminates lost updates. Under what circumstances will performance be improved as compared with running Trans as shown in the previous question at REPEATABLE READ?

Solution

The instance of Trans that executes its UPDATE last will find that the value of attrib is different from the value returned by its SELECT statement. Since the second condition in its WHERE clause is not satisfied, its UPDATE will have no effect. Thus Trans should check if its UPDATE has modified a row and abort if no row has been updated. A performance improvement is possible if the interleaving of two instances of Trans is unlikely since read locks are released early. For example, a transaction that updates an attribute other than attrib requires a write lock on the row. This can't be granted at REPEATABLE READ, but it can at READ COMMITTED. Such a transaction does not cause a lost update and therefore a potential for performance improvement exists with the new UPDATE statement.

(c) Are lost updates possible with SNAPSHOT isolation? Explain your answer by showing how they are either prevented or allowed. If your answer here relates to the previous part of this question explain how.

Solution

The SNAPSHOT concurrency control determines that the second instance of Trans to request commit must be aborted since the value of attrib has been changed since it executed its SELECT statement. Hence the concurrency control is making essentially the same check that was made by the UPDATE statement in the previous part.

8. A schema has two tables:

Student (Id, Name, .) - Id and Name are both unique
Registered (Id, CrsCode, Credit) - contains one row for each course the student is taking this semester.

The tables are accessed by a transaction, T, having two SQL statements:

```
SELECT SUM (R.Credits), S.Id          UPDATE Registered
INTO :sum , :id                       SET Credits = Credits + 1
FROM Student S, Registered R          WHERE Id = :id AND CrsCode = :crs
WHERE S.Name = 'Joe' AND S.Id = R.Id
GROUP BY S.Name, S.Id
     (1)                                   (2)
```

T consists of statement (1) followed by statement (2) (and local computation between them). (1) returns the number of credits for which Joe is registered together with his Id. T maintains the integrity constraint "no student shall register for more than 20 credits". If Joe has less than 20 credits, T executes (2) to increment the number of credits for which Joe has registered in one (the particular one is not specified) of his courses.

Suppose Joe executes two instances of T concurrently at the following isolation levels. In each case say whether or not the named violation can occur and if the answer is yes, explain how (e.g., what locks are or are not held) below. (Answers are shown in parenthesis.)

(a) READ COMMITTED
Lost update (no)
Violation of the integrity constraint (yes)
Deadlock (no)

(T_1 and T_2 both get short term shared locks on all Joe's rows in Registered. Then T_1 gets a long term exclusive lock on one row,increments credits and commits. T_2 then does the same on a [possibly the same] row.)

(b) REPEATABLE READ
Lost update (no)
Violation of the integrity constraint (no)
Deadlock (yes)

(T_1 and T_2 both get long term shared locks on all Joe's rows in Registered. Then T_1 requests an exclusive lock on one of those rows and must wait. Then T_2 requests an exclusive lock on one of those rows and also waits.)

(c) SNAPSHOT
Lost update (no)
Violation of the integrity constraint (yes)
Deadlock (no)

(T_1 and T_2 both execute (1) from the same version. Then T_1 executes (2) using *course*1 as the value of :crs and T_2 executes (2) using a different value of :crs.)

9. Give examples of serializable schedules that would be accepted

   (a) At SNAPSHOT isolation but not at REPEATABLE READ
   Solution:
   $r_1(x)\ r_2(x)\ w_2(x)\ commit_2\ w_1(y)\ commit_1$

   (b) At SNAPSHOT isolation but not by strict two-phase locking control
   Solution:
   $r_1(x)\ r_2(x)\ w_2(x)\ commit_2\ w_1(y)\ commit_1$

   (c) At SERIALIZABLE but not at SNAPSHOT isolation
   Solution:
   $r_1(x)\ r_2(y)\ w_1(x)\ commit_1\ r_1(x)\ w_2(x)\ commit_2$

   (d) By a strict two-phase locking control, but not by a timestamp ordered control.
   Solution:
   $r_1(x)\ r_2(x)\ commit_2\ w_2(x)\ commit_1$

   (e) By an optimistic control, but not by a strict two-phase locking control.
   Solution:
   $r_1(x)\ r_2(x)\ w_2(x)\ commit_2\ commit_1$

10. Show that if a schedule produced at SNAPSHOT isolation has the property that whenever a transaction reads some item it also writes that item, then that schedule is serializable.

    Solution:

    Consider any two transactions that are executing concurrently. Since their writes must be disjoint and every item that each transaction reads it also writes, all accesses of the two transactions must be disjoint. Therefore they can be serialized in either order. There are never any conflicts between concurrent transactions.

Problems for Chapter 22: Atomicity and Durability

1. Short answer questions

   (a) What action causes a transaction to become durable?

   Solution

   the commit record is written to the log on maas store

   (b) With mirrored disks the database is replicated on distinct mass storage devices by writing to both devices whenever the database is updated. It is assumed that the disks never fail at the same time. What is the major advantage of using mirrored disks? Is it necessary to have a write-ahead log if mirrored disks are used? Explain.

   Solution

   Advantage - increased availability after media failure
   A write-ahead log is still required in order to roll back active transactions or roll forward committed transactions after a crash. The before and after images in update records are needed to do this.

   (c) Must the log buffer be flushed when an abort record is appended to it? Explain.

   Solution

   no - if the system crashes before the abort record is moved to mass store, the transaction will be aborted anyway.

   (d) Suppose that a dirty page in the cache has been written by two active transactions and one of them commits. Assume that log records contain both before and after images. Are there any constraints of the form ”the page cannot be written to mass store until x”? Are there any constraints of the form ”the page must be written to mass store before y”? If such constraints exist explain x or y. If not, justify your answer.

   Solution

   - The page cannot be written until both update log records have been flushed to preserve the write-ahead property.
   - There is no restriction on when the page must be written since update records contain after images. If the system crashes the effect of committed transactions can be constructed from the after images. (Some students might write an answer specifying a constraint related to the writing of a checkpoint record. This is a better answer.)

2. A checkpoint record in the log contains a list of transaction Ids.

   (a) What causes a transaction Id to be placed in the list?

   Solution

   The transaction is active at the time the checkpoint record is appended to log.

   (b) Suppose the system crashes and *tid* is on the list contained in the last checkpoint record in the log.

      i. What condition has to be satisfied in order for *tid* to represent a transaction that was active at the time of the crash?
      Solution
      No commit or abort record for *tid* follows the checkpoint record in the log.

      ii. Is it possible that a transaction active at the time of a crash is not named on the list? Explain.
      Solution
      Yes. The transaction's begin record follows the checkpoint record and no commit or abort record for that transaction follows the begin record.

   (c) Checkpoint records play a role in pass 2 of the three-pass algorithm used for crash recovery when a no-force policy is used.

      i. What is the purpose of pass 2?
      Solution
      Ensure that all updates executed before the crash are contained in the database.

      ii. What is the pass 2 algorithm that achieves that purpose?
      Solution
      Scan log forward from checkpoint record and use after images in each update record to update the database.

      iii. What condition must be satisfied before a checkpoint record is placed in the log in order to guarantee that the algorithm works?
      Solution
      All dirty pages in cache are flushed before checkpoint record is appended.

3. (a) What is the meaning of the log sequence number (LSN) associated with a database page in the cache?

   Solution (see text)

   (b) How is it used to enforce the write-ahead property?

   Solution (see text)

   (c) Assume physical logging and a no-force policy. Does the LSN associated with a page have to be stored *in* the page? Explain.

   Solution

   No. Physical updates are idempotent. Hence, on crash recovery it is not necessary to know whether a physical update described in an update record has already been applied to an item in a database page. The update can simply be applied since multiple applications yield the same result.

   (d) Assume logical logging and a no-force policy. Does the LSN associated with a page have to be stored *in* the page? Explain.

   Solution

   Yes. Logical updates are not idempotent. Hence, on crash recovery it is necessary to know whether a logical update described in an update record has already been applied to an item in a database page.

4. Some systems have demanding performance and availability requirements. Do the following play a role in enhancing the performance of a system? Do they enhance availability? Explain your answers.

   (a) page cache

   Solution

   enhances performance since page I/O is reduced
   not availability since cache is destroyed in a crash

   (b) log buffer

   Solution

   enhances performance since log I/O is reduced
   not availability since buffer is destroyed in crash

   (c) checkpoint record

   Solution

   enhancse availability since recovery is speeded
   not performance - (if anything it reduces performance since additional log records must be written)

   (d) physiological logging

   Solution

   enhances performance - reducing the size of log records reduces log I/O
   doesn’t affect availability

5. A DBMS crashes and on recovery finds the log in the state shown in Figure 5

begin T_1	update T_1 x bef 0 aft 5	begin T_2	update T_2 y bef 0 aft 7	begin T_3	commit T_2	check T_1 T_3

begin T_4	update T_3 y bef 7 aft 8	commit T_1	update T_4 z bef 0 aft 3

Figure 4: Log on recovery from crash.

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

Solution

T_3 and T_4

(b) What were the initial values of x, y, and z?

Solution

0, 0, 0

(c) What are the values of x, y, and z after pass 2 of the three pass recovery procedure?

Solution

5, 8, 3

(d) What are the values of x, y, and z after pass 3 of the three pass recovery procedure?

Solution

5, 7, 0

6. (a) A DBMS uses a page cache, a log buffer, no-force commit processing, and fuzzy checkpoints. For each of the three passes of the recovery procedure state the purpose of the pass (do not describe the algorithm) and the portion of the log that must be scanned.

Solution

Pass 1: purpose: (determine which transactions are active at time of crash)
portion of log: (log end to last checkpoint record in log)

Pass 2: purpose: (roll forward the database)
portion of log: (penultimate checkpoint record to log end)

Pass 3: purpose: (roll back active transactions identified in pass 1)
portion of log: (log end to begin record of oldest transaction identified in pass 1)

(b) Suppose, in pass 1 of the above recovery procedure it is found that the most recent checkpoint record, CK, is empty. Does the log scan in pass 3 stop before CK or after CK? Explain.

Solution

Before CK. Since all active transactions started after CK was written

(c) What is the difference between the contents of an update record in the log when a force or a no-force policy is used for commit processing?

Solution

No-force policy writes after images in update record.

(d) A force policy has the potential of requiring forced writes at three points during processing of a transaction's commit.. What is written by each and why must the write be forced?

Solution

(1) Write all the transaction's update records remaining in the log buffer (to maintain the write-ahead policy).
(2) Write all the transaction's dirty pages in the cache (to ensure durability).
(3) Write the transaction's commit record (to commit the transaction).

(e) Hotspots are not handled efficiently in the cache when a force policy is used. (A hotspot is a database page that is frequently updated.) Explain why this is so.

Solution

Because each dirty page must be written when a transaction commits, the page containing the hotspot must be written when each transaction that accesses it commits. As a result, the cache is not used efficiently, since the intent is that actively used pages need not be flushed from the cache.

Problems for Chapter 23: Architecture of Transaction Processing Systems

1. Short answer questions

   (a) Two steps are involved in preparing to execute a remote procedure call.

      i. The interface to the procedure, written in an interface definition language, must be compiled. What two things are produced (just name them)?
      Solution
      header file, client stub

      ii. The location of the server that will execute the procedure cannot be determined until run time. How is this done? (Two sentences, at most, indicating that you know what is going on will answer this.)
      Solution
      The server registers its address with the Directory Server, the client stub queries the Directory Server and requests the server's address

   (b) The tx interface is defined for interaction between an application and a transaction manager and the xa interface is defined for interaction between a resource manager and the transaction manager.

      i. Which of the ACID properties are these interfaces (and the transaction manager) designed to support?
      Solution
      atomicity

      ii. Name one method of the tx interface. What are its parameters and what does it do?
      Solution
      tx_begin - has a transaction Id as an out parameter. It registers a new distributed transaction with the transaction manager or

      tx_commit - has the transaction Id as an in parameter and returns an indication of success or failure. Its purpose is to commit a distributed transaction

      iii. Name one method of the xa interface. What are its parameters and what does the method do?
      Solution
      xa_reg - It has a transaction Id and a server name as an in parameter. Its purpose is to join the server as a cohort to the named distributed transaction.

   (c) Sessions are set up between communication entities when peer-to-peer communication is used.

      i. What is meant by context?
      Solution
      Information describing the state of a process participating in a session.

      ii. What is a context handle?
      Solution
      Data structure stored at one peer, P, in a session with P' containing either a pointer to the session's context used by and stored at P' or containing P''s context. The

context handle is passed with each message sent by P to P' and returned by P' to P with the response.

2. State which one or more of the Java Enterprise Beans (session beans, stateless session beans, message-driven beans, entity beans) are characterized by each of the following properties

   (a) It must execute within a transaction
   Solution:
   entity bean

   (b) It cannot execute within a transaction started by the client that called it
   Solution:
   message-driven bean

   (c) It cannot have bean-managed transactions
   Solution:
   entity bean

   (d) Changes made to the bean are persistent in that they are propagated to the database
   Solution:
   entity bean

   (e) Communication with the bean by the calling client is asynchronous
   Solution:
   message-driven bean

Problems for Chapter 24: Implementing Distributed Transactions

1. Short answer questions

   (a) In the two-phase commit protocol the cohort forces a prepare record to its log before sending a ready vote. What bad thing might happen if the record was not forced?

   Solution

   If a ready vote is sent and the cohort crashes before the prepare record is written it may be the case that all update records are not durable at the cohort but the coordinator commits the transaction. In this case the cohort will not be able to commit.

   (b) A cohort can be blocked during the execution of the two-phase commit protocol. Explain what this means and why it is bad.

   Solution

   Cohort has to wait until it determines whether the coordinator has committed or aborted the transaction. Bad because cohort can't release the subtransaction's locks for an indeterminate period of time

   (c) Even though all subtransactions at each site in a distributed database are executed at SERIALIZABLE, distributed transactions that invoke these subtransactions might not be serializable. Explain how this can happen.

   Solution

   T_{1A}, T_{1B} are subtransactions of T_1 at sites A and B
   T_{2A}, T_{2B} are subtransactions of T_2 at sites A and B
   T_{1A} has an operation that conflicts with and precedes an operation of T_{2A}
   T_{2B} has an operation that conflicts with and precedes an operation of T_{1B}

   (d) With primary copy (asynchronous-update) replication

      i. Which replica is locked and read when a transaction executes a read request?
      Solution
      Any replica

      ii. Which replica is locked and updated when a transaction executes a write request?
      Solution
      the primary copy

      iii. Explain how updates are propagated to all replicas.
      Solution
      After the transaction commits the new value of the primary copy is transmitted from the primary site to all secondary sites.

      iv. Give an example that illustrates the non-serializable behavior that can result with this form of replication even though locks are released in a strict, two-phase fashion.
      Solution

      T_1: $w(x_{pri})$ $w(y_{pri})$ *commit*
      T_2: $r(x_{sec})$ $r(y_{pri})$ *commit*
      $T_{update-rep}$: $w(x_{sec})$ $w(y_{sec})$

(e) Suppose for each subtransaction of a distributed transaction there exists a compensating subtransaction and the two-phase commit protocol is modified so that the cohort can either commit or abort the subtransaction before sending a vote message to the coordinator. If the cohort commits the subtransaction and the coordinator responds with an abort message in phase 2, the cohort runs the compensating subtransaction. Does the protocol provide global atomicity? Explain.

Solution

No A concurrent transaction might see the result of a committed subtransaction before compensation is applied.

2. (a) If all sites in a distributed database system use strict two-phase locking (*no* early lock release) and two-phase commit, then distributed transactions are (globally) serializable. To demonstrate that you understand the argument that supports this statement explain why, if an operation of a subtransaction of $T1$ at some site conflicts with and precedes an operation of a subtransaction of $T2$ at that site (and if both transactions commit), then $T1$ commits (globally) before $T2$.

   Solution

   If an operation of $T1_A$ conflicts with an operation of $T2_A$ and $T1$'s operation comes first then $T1_A$ has acquired a lock for which $T2_A$ is waiting. Since in a two-phase commit subtransactions don't release locks until the transaction commits, $T1_A$ doesn't allow $T2_A$ to continue until after $T1_A$ commits. Hence, $T2$ can't commit before $T1$.

   (b) Suppose, instead of strict two-phase locking, the concurrency control at each site is two-phase, but early release of read (but not write) locks is allowed. Assuming two-phase commit is used, will distributed transactions be globally two-phase commit is used, will distributed transactions be globally serializable? Explain your answer.

   Solution

   Not necessarily. $T1_A$ might release a read lock on x and $T2_A$ might then acquire a write lock on x. Hence, $T1_A$ and $T2_A$ conflict. A similar conflict might occur in the opposite direction at site B. Since this involves no waiting, the two transactions can both commit.

3. (a) What is the difference between two-phase locking and strict two-phase locking?

Solution

All locks are held until transaction commits in strict two-phase.
Read locks can be released early (but in a two-phase fashion) if the concurrency control is not strict.

(b) If, in a multidatabase system, all concurrency controls use a strict two-phase locking protocol and a two-phase commit protocol is used to achieve global atomicity then global transactions are serializable. Does this result still hold if the concurrency controls are two-phase, but not strict? Explain.

Solution

No. T_{1A} can release a read lock on x and T_{2A} can then write x. Hence $T_1 \rightarrow T_2$. The same thing can happen with another database item at B, except in the opposite order with the result that $T_2 \rightarrow T_1$. Both transactions can commit, but they will not be globally serializable.

4. (a) If each cohort site in a network uses two-phase locking to implement serializable schedules locally and a two-phase commit protocol is used to implement global atomicity, then transactions are globally serializable. Sketch a proof that demonstrates this.

   Solution

   see proof in Section 24.5 of text

   (b) A forced write is used at three points in the two-phase commit protocol with presumed abort. Explain why this is necessary by showing the problem that results if any of the writes is not forced.

   i. The cohort forces a prepare record before sending a vote message.
   Solution
   - cohort votes ready but crashes before prepare record is durable.
   - cohort aborts on restart.
   - coordinator commits the transaction.

   ii. The coordinator forces a commit record before sending the commit message.
   Solution
   - coordinator sends commit to all cohorts but crashes before its commit record is durable.
   - coordinator has no record of transaction on restart.
   - some cohorts commit.
   - some cohorts (in their uncertain period) don't get commit message and query the coordinator. Coordinator will respond abort since (using the presumed abort property) it has no record of the transaction.

   iii. The cohort forces a commit record before sending a done message.
   Solution
   - cohort crashes after sending done message but before its commit record is durable.
   - coordinator deletes transaction record when done messages are received from all cohorts.
   - cohort is in prepared state on recovery and queries the coordinator, who responds abort (using the presumed abort property).

5. (a) With primary copy replication:

   i. What happens when a transaction requests to read an item?
   Solution
   It gets a read lock on the nearest replica and reads it.

   ii. What happens when a transaction requests to write an item?
   Solution
   It gets a write lock on the primary copy and updates it.

   (b) Assuming transactions are executed at the SERIALIZABLE isolation level:

   i. Can a dirty read occur with primary copy replication? Explain.
   Solution
   no - only committed values can be read

   ii. Can a dirty write occur with primary copy replication? Explain.
   Solution
   no - a write can't occur until the write lock on the primary is released, and hence only occurs when the writer commits.

   iii. Can a lost update occur with primary copy replication? Explain.
   Solution
   yes - $T1$ and $T2$ can read lock and read distinct replicas and then successfully write lock and write the primary and commit.

6. (a) Define mutual consistency and weak mutual consistency in a replicated system.

Solution

mutual consistency: all replicas identical when an update transaction completes
weak mutual consistency: all replicas will eventually become identical

(b) Group replication is a form of asynchronous replication that does not support weak mutual consistency. Give an example that demonstrates this problem.

Solution

Assuming that there are three replicas of x, the following schedule illustrates that the interleaved execution of two transactions can cause a violation of weak mutual consistency since the final values of x_1 and x_3 are produced by T_2 while the final value of x_2 is produced by T_2

$$w_1(x_1)\ w_2(x_2)\ w_1(x_3)\ w_2(x_3)\ w_1(x_2)\ w_2(x_1)$$

(c) A conflict resolution strategy is used with group replication to ensure weak mutual consistency. Describe one such strategy.

Solution

Each transaction has a timestamp. Each replica has the timestamp of the last transaction to update it. Apply update only if the timestamp of the updating transaction is greater than the timestamp of the replica it is updating.

(d) Weak mutual consistency is not a problem with primary copy replication (another form of asynchronous replication). Explain why this is so.

Solution

All transactions that update x must lock the primary copy. Replicas updated from primary in the order that the primary is updated.

7. With the linear commit protocol, sites are connected in a chain, and the subtransaction at the left-hand end of the chain initiates the protocol. Propose a crash recovery protocol for a site in the chain. Consider the following three points at which the crash might occur and state in each case (1) how the site determines from its log that it is at that point and (2) what action it takes (beyond standard recovery processing)

   (a) The site crashes before the first message from its left-hand neighbor arrives.

   Solution

   (1) no prepare record

   (2) no action beyond standard crash recovery

   (b) The site crashes after sending the first message to its neighbor on the right, but before receiving a message from its neighbor on the right.

   Solution

   (1) prepare record in log, but no commit record

   (2) poll neighbors for status of transaction and block until answer is determined

   (c) The site crashes after sending a message to its neighbor on the left but before receiving the second message from its neighbor on the left.

   Solution

   (1) commit record in log, but no complete record

   (2) reconstruct transaction record

   Alternate solution:

   (1) abort record in log

   (2) no action beyond standard crash recovery

8. We wish to design a revised two-phase commit protocol in which cohorts do not have to send a done message after they have committed. The new protocol will not have the presumed-abort property, so the coordinator will have to consult its log under some circumstances. The protocol for the cohorts is exactly the same as for the presumed-abort protocol, except that cohorts that commit do not have to send a done message after they have committed. The coordinator maintains a transaction record in its main memory. Explain what the coordinator does under the following circumstances

   (a) When it commits the transaction

   Solution:

   Forces a commit record to its log and sends a commit message to each cohort.

   (b) When it aborts the transaction

   Solution:

   Sends an abort message to the cohorts

   (c) When it restarts after it crashes

   Solution:

   Nothing

   (d) When it is asked by a cohort what the state of the transaction is

   Solution:

   If it finds a transaction record in its main memory, it replies with the information in the record.

   If it does not find a transaction record in its main memory, it looks in its log. If it finds a commit record, it replies commit; if it finds nothing for that transaction it replies abort

9. We are building an application that consists of a distributed nested transaction in which each child executes at a different distributed site (and a different site from its parent). We would like to optimize the application in two different ways.

   (a) Describe how the locking protocol can be simplified compared to the locking protocol for nested transactions described in the text

   Solution:

   A child does not have to request locks from its parent or return locks to its parent when it has completed. It can just obtain the locks that it needs and keep them until the overall transaction commits or aborts.

   (b) Describe how the two-phase commit protocol can be simplified compared to that two-phase commit protocol described in the the text (so that fewer messages are required in the protocol).

   Solution:

   When a child (conditionally) commits, it goes into a prepared state. The first phase of the commit procedure is not needed. The coordinator just sends a commit message to each child, which replies with a done message.

Problems for Chapter 25: Web Services

1. The chapter introduces a number of related concepts: port type, port, binding, message, variable, service, endpoint reference, operation. Give a description (several sentences) of each of the above concepts stating what it is and what relationship it has with other concepts.

   Solution

   (a) port type - (WSDL) specifies a set of related operations to be exported as a unit.

   (b) operation - (WSDL) specifies input and output messages that are used in invoking the operation; it is a child of a portType

   (c) message (WSDL) specifies the types of each field in the message; child of message

   (d) variable - (BPEL) specifies a variable of type message for holding a message

   (e) service - (WSDL) specifies a set of related ports

   (f) port - (WSDL) associates a network address with a binding; child of a service

   (g) binding - (WSDL) specifies how to invoke operations of a particular portType usng a particular protocol

   (h) endpoint reference - (WS Addressing) specifies all the information needed to send a message to a particular port type

2. SOAP extensibility relates to the use of intermediaries and SOAP headers to add features to a web service that is invoked using SOAP messaging. Give an example of an extension and how an intermediary and header would be used to implement it.

   Solution

   Add authentication. Message would contain a message body and an authentication header containing a name, password and ultimate destination. Message would be addressed to an authentication proxy. Proxy would delete the header, do authentication based on name and password, add a new header containing authenticated Id and relay new header and original body to ultimate destination.

3. A web service exports an interface containing a single, asynchronous operation using a message that carries a document that is an instance of a schema element with name `persons`. In the following parts be clear about your use of namespaces and prefixs.

   (a) Give a WSDL declaration of the message.

   Solution

```
<message name=''R''>
       <part name=''L'' element=''M:persons''/>
<message>
```

   `M` is the prefix assigned to the target namespace of the schema document (the document containing the declaration of the element `persons`.).

   (b) Give a WSDL declaration of the interface.

   Solution

```
<portType name=''Z''>
       <operation name=''W''/>
              <input message=''X:R'' />
       </operation>
</portType>
```

   `X` is the target namespace of the WSDL document (the document containing this and the message declarations).

   (c) Assuming the binding is document/literal using SOAP over HTTP, give the data part of the HTTP message. You need not show the HTTP header nor the details of an instance document. Just show how the instance is embedded in the SOAP format.

   Solution

```
<s:Envelope xmlns:s=''P''>
       <s:Body>
              <persons xmlns=''Y''>
                     ...
              </persons>
       </s:Body>
</s:Envelope>
```

   P is the SOAP envelope namespace; Y is the prefix assigned to the target namespace of the schema document (the document containing the declaration of the element `persons`.).

   (d) Suppose the web service accepts a persons document, produces a new persons document and returns it to the client via a callback. Give the additional WSDL declarations that would be needed and a BPEL `partnerLinkType` declaration.

   Solution

```
<portType name=''A''>
       <operation name=''B''/>
              <input message=''X:R'' />
       </operation>
</portType>

<partnerLinkType name=''D''>
       <role name=''client''>
              <portType name=''X:A''/>
       </role>
       <role name=''server''>
              <portType name=''X:Z''/>
       </role>
</partnerLinkType>
```

4. Two business process, $P1$ and $P2$, communicate. $P1$ invokes operations in port type $I21$ and $I22$ supported in $P2$ and $P2$ invokes operations in port type $I1$ supported in $P1$.

   (a) i. Specify all partner link type declarations.

   Solution

```
<partnerLinkType name=''PLT1''>
       <role name=''N1''>
              <portType name=''I1''/>
       </role'>
       <role name=''N2''>
              <portType name=''I21''/>
       </role'>
</partnerLinkType>

<partnerLinkType name=''PLT2''>
       <role name=''N3''>
              <portType name=''I22''/>
       </role'>
       <role name=''N4''>
              <portType name=''I1''/>
       </role'>
</partnerLinkType>
```

   ii. Specify all partner link declarations in each process (and specify which process contains the declaration).

   Solution
   In process $P1$

```
<partnerLink name=''PL3'' partnerLinkType=''PLT1''
       myRole=''N1'' partnerRole=''N2''/>
<partnerLink name=''PL4'' partnerLinkType=''PLT2''
       myRole=''N4'' partnerRole=''N3''/>
```

   In process $P2$

```
<partnerLink name=''PL1'' partnerLinkType=''PLT1''
       myRole=''N2'' partnerRole=''N1''/>
<partnerLink name=''PL2'' partnerLinkType=''PLT2''
       myRole=''N3'' partnerRole=''N4''/>
```

iii. Specify all partners declarations in each process (and specify which process contains the declaration).

Solution
In $P1$

```
<partners name=''XX''>
        <partnerLink name=''PL3''/>
        <partnerLink name=''PL4''/>
</partners>
```

(b) Suppose the conversation between $P1$ and $P2$ is initiated when $P1$ invokes an operation, $O2$ of interface $I21$ in $P2$ using a message of type $M2$. A new instance of $P2$ is to be created to manage a stateful interaction with $P1$. Assuming $M2$ contains a part *clientId* of type string, give the following:

i. property declaration

Solution

```
<property name=''P'' type=''string''/>
```

ii. property alias declaration

Solution

```
<propertyAlias propertyName=''P'' messageType=''M2''
        part=''clientId''/>
```

iii. correlation set declaration

Solution

```
<correlationSet name=''C'' properties=''P''/>
```

iv. the receive statement in $P2$ the invocation.

Solution

```
<receive name=''UU'' partnerLink=''PL1'' portType=''I2''
        operation=''O2'' variable=''V2'' createInstance=''yes''>
        <correlations>
                <correlation set=''C'' initiate=''yes''/>
        </correlations>
</receive>
```

5. A stock brokerage company offers a getQuote service in which a requestor sends a getQuoteRequest message containing a stock symbol and the brokerage company responds with a GetQuoteResponse message containing the current quotation for that stock. Information about the service is stored in a UDDI registry.

   (a) Explain in which of the UDDI data structures the following information about the service is stored:

      i. The name of the service
      ii. The Web address at which the service is provided
      iii. The WSDL portType description of the service
      iv. The binding portions of the WSDL description of the service.

      Solution

      i. The name is stored in the businessService
      ii. The Web address is stored in the bindingTemplate
      iii. The portType description is stored in a tModel
      iv. The binding portions of the WSDL description is stored in a tModel.

   (b) Suppose that all I know is the name of the brokerage company. Describe informally the sequence of UDDI queries I would use to find out the details of the SOAP messages needed to invoke the service

      Solution

      i. A find_business query using the company name returns the appropriate businessEntity, which contains the serviceKey
      ii. A get_serviceDetail query using the serviceKey returns the appropriate businessService, which contains the bindingKey
      iii. A get_bindingDetail query using the bindingKey returns the appropriate bindingTemplate, which contains tModelKeys
      iv. A get_tModelDetail query using the tModelKeys returns the appropriate tModels, which include the needed information about the SOAP messages needed to invoke the service.

6. The distinction between durable and volatile two-phase commit in the WS-AtomicTransaction protocol is concerned with the prepare message. What is the difference between the protocols? Explain your answer in terms of the example used in class.

   Solution

   Prepare messages for cohorts registered for durable two-phase commit are not sent by the coordinator until all votes from all cohorts who have registered for volatile two-phase commit are received.

   This allows volatile cohorts that are caching data obtained from durable sites to flush dirty pages back to those sites. Hence, durable cohorts can write all dirty pages before responding to coordinator with vote.

Problems for Chapter 26: Security and Electronic Commerce

1. The following questions relate to digital signatures.

   (a) What is a message digest function, f?

   Solution

   Mapping of a message M to a small, fixed size bit string.

   (b) Explain what a one-way function is and why the message digest function must be one-way.

   Solution

   A function, f is one-way if it is easy to compute $f(M)$, but hard to find an M such that $f(M)$ is equal to some given value.

   (c) A digital signature appended to a message M authenticates the sender and guarantees the integrity of M.

      i. Explain what a digital signature on M by a process P is.
      Solution
      $K_P^{pri}[f(M)]$; the message digest of M encrypted with P's private key.

      ii. What steps does the receiver take on receiving the signed message?
      Solution
      The message sent is $(M,\ K_P^{pri}[f(M)])$
      The receiver computes $f(M)$, decrypts the signature using K_P^{pri} and compares the decrypted value with the computed value.

      iii. How does this prove to the receiver that P constructed the signed message?
      Solution
      Only P can have constructed the signature.

      iv. How does this guarantee the integrity of M?
      Solution
      If M is changed to M' then the computation at the receiver yields $f(M')$, which will not match the value $f(M)$ decrypted by the receiver.

2. (a) What is a certificate (explain the fields it contains)? What security problem does it solve? How does it solve this problem? (Three answers are required for full credit.)

Solution

fields: (name, URL, public key) signed with private key of a Certification Authority
problem: key distribution
solution: CA is a trusted third party. Its signature guarantees the association among the (name, URL,public key). A client can encrypt a message using the contained public key and send it to the URL with confidence that only the named entity will be able to decrypt the message.

(b) Using Kerberos, client C obtains a ticket from key server KS for use with server S. The ticket contains C's name. Describe an attack that intruder I can successfully use if this field were eliminated from the ticket.

Solution

- I obtains ticket containing K_{sess} from KS for use with S
- I constructs authenticator using K_{sess} containing C's name
- Since C's name is not contained in the ticket, S cannot tell that the message is not coming from C - S performs service thinking it wsa requested by C

3. Suppose S wants to send some binary data, D (an arbitrary bit string), to R. Hence, when R extracts the data from the message it receives it will have no way of testing the data to see if it makes sense. Furthermore, we assume that R has no way of communicating back to S to validate the message. Assume that an intruder, I, can copy messages off the net and change or replace messages that have been transmitted, but cannot steal keys. We require that (1) R not be fooled into accepting incorrect data and (2) D is not revealed to I. For each of the following encryption schemes say "yes" if the requirement is satisfied and "no" otherwise (no explanation is required).

   (a) Assuming S and R have agreed on a (symmetric) session key, K_{sess}, S sends $K_{sess}[D]$.
   (1) no (I sends arbitrary bit string that is decoded to a D')
   (2) yes

   (b) Assuming S knows R's public key, K_r^{pub}, S sends $K_r^{pub}[D]$.
   (1) no (same problem as previous case)
   (2) yes

   (c) Assuming S and R have agreed on a message digest function, f, S sends $(D,\ f(D))$.
   (1) no (I sends $(D'\ f(D'))$
   (2) no

   (d) Assuming S and R have agreed on a message digest function, f, and R knows S's public key, K_s^{pub}, S sends $(D,\ K_s^{priv}[f(D)])$.
   (1) yes
   (2) no

   (e) Assuming S and R have agreed on a message digest function, f, and a session key, K_{sess}, and that R knows S's public key, K_s^{pub}, S sends $K_{sess}[(D,\ K_s^{priv}[f(D)])]$.
   (1) yes
   (2) yes

   (f) f) Assuming S and R have agreed on a message digest function, f, and a session key, K_{sess} , and that R knows S's public key, K_s^{pub}, S sends $(K_{sess}[D],\ K_s^{priv}[f(D)])$.
   (1) yes
   (2) yes

   (g) Assuming S and R have agreed on a message digest function, f, and a session key, K_{sess}, S sends $(K_{sess}[D],\ f(D))$.
   (1) yes
   (2) yes

   (h) Assuming S and R have agreed on a message digest function, f, and a session key, K_{sess}, S sends $K_{sess}[D, f(D)]$.
   (1) yes
   (2) yes

   (i) Assuming S and R have agreed on a message digest function, f, and a session key, K_{sess}, S sends $(K_{sess}[D],\ f(K_{sess}[D]))$.
   (1) no (I sends $(x,\ f(x))$ and R retrieves D' from x
   (2) yes

4. (a) The body of a SOAP message is to be signed using XML Signature. The < *signature* > element has a number of fields, some of which are contained in a < $signedInfo$ > child. Give a brief explanation (one or two sentence should suffice) of the purpose of each of the following and indicate whether or not they are contained in < $signedInfo$ >.

   i. Canonicalization method

      Solution
      put document in a standard format so two documents with different space, tab, etc characters yield the same signature (in <signedInfo>)

   ii. Signature method

      Solution
      method used to sign <signedInfo> (in <signedInfo>)

   iii. Digest method

      Solution
      method used to digest the body (in <signedInfo>)

   iv. Digest value

      Solution
      result of digesting the the body using the digest method (in <signedInfo>)

   v. Signature value

      Solution
      result of signing <signedInfo> using signature method (not in <signedInfo>)

   (b) How is the signature element included in the SOAP message? (This is a question about WS-Security and a one or two sentence answer suffices.)

      Solution
      stored in header described by WS-Security

5. What happens if

   (a) In the Kerberos protocol when the customer, C, sends the first message to the key server (in the clear) asking for a ticket to authenticate C to a particular server, S, an intruder intercepts the message and instead sends to the key server a message asking for a ticket to authenticate C to itself, S'.

      Solution:

      The procedure proceeds as usual. The key server generates a session key and returns to the customer a message containing two parts, the second of which is the ticket. However the first part of that message, which can be decoded by the customer, contains the name of the intruder server, S' instead of the requested server. The customer must use this information to determine that the intrusion has taken place.

   (b) In the SSL protocol when the customer asks the server for its certificate, an intruder intercepts that message and sends to the customer its own (valid) certificate instead.

Solution:

The procedure proceeds as usual. The customer's browser accepts the certificate, generates a session key, and sends it to the intruder. The interaction can then continue. To detect the intruder, the customer would have to manually examine the certificate to determine that the certificate belongs to the intruder and not the requested server.

(c) In the SET protocol, after the customer sends to the merchant a two-part message, one of which is encrypted with the merchant's public key and the other with the the payment gateway's public key, an intruder intercepts that message and substitutes its own version of the two parts, each encrypted with the appropriate key.

Solution: Since the intruder does now have the customer's private key, it cannot perform the dual signature of the new message, and therefore the customer will detect the intruder.

(d) In the Ecash protocol, after the customer receives freshly minted tokens from the bank, it claims that the tokens were never received and asks the bank for additional copies of the tokens.

Solution: The bank can send the copies. Since each token has a unique serial number and, when someone tries to spend a token, the bank checks for duplicate serial numbers, the customer cannot spend both copies of the token.

Problems for Appendix A: An Overview of Transaction Processing

1. Give schedules that demonstrate incorrect executions in each of the following situations. For each situation, give a one sentence explanation of why the execution is incorrect (for example, 'The balance in the bank account is incorrect because ...")

   (a) Transactions are not isolated

   Solution:

   $r_1(bal = 10)\ \ r_2(bal = 10)\ \ w_1(bal = 110)\ \ w_2(bal = 15)\ \ c_1\ \ c_2$

   Two deposit transaction. The first deposits \$100; the second \$5. The final balance is incorrect. The effect of the first deposit has been lost

   (b) Transactions are not atomic

   Solution:

   $w_1(x = 100)\ \ r_2(x = 100)\ \ w_2(y = 100)\ \ c_2\ \ a_1$

   The second transaction was trying to copy the value of x to y, but it copied a value that was later rolled back. Thus the first transaction had an effect even though it did not commit.

2. Give an example of non-serializable schedules that can be produced at the READ UNCOMMITTED and READ COMMITTED isolation levels

   Solution:

   (a) READ UNCOMMITTED

   $w_1(x)\ r\_2(x)\ r_2(y)\ w_2(y)\ c_1\ c_2$

   (b) READ COMMITTED

   $r_(x)\ w_2(x)\ w_2(y)\ c_2\ r_2(y)\ c_1$

3. Give an example of a serializable schedule that could be produced at SNAPSHOT isolation, but could not be produced by a strict two-phase locking concurrency control.

   Solution:

   $r_1(x)\ \ w_1(x)\ \ r_2(x)\ \ r_2(y)\ \ w_2(y)\ \ r_1(y)\ \ c_1\ \ c_2$

4. Explain how granular locking can be used at the SERIALIZABLE isolation level.

   Solution:

   If a transaction wants to read one or more tuples in a table, it gets an S lock on the entire table. If a transaction wants to write one or more tuples in a table it gets a SIX lock on the table and an X lock on the tuples it writes. No phantoms can occur because if one transaction has an S lock on the table, no other transaction can get a SIX lock on that table.

5. Explain each of the following.

   (a) Why a log has to have the write-ahead property

   Solution:

   If the information was written to the database first and then to the log, and the system crashed between the two writes, there would be no way to rollback the changes

   (b) Why a checkpoint record is appended to the log periodically

   Solution:

   Otherwise, in the event of a crash, the entire log would have to be scanned to perform the recovery

6. In the two-phase commit protocol, suppose a cohort crashes in each of the following situations. Explain what should it do when it recovers

   (a) It crashed before it sends it vote message

   Solution:

   Abort. It knows the coordinator could not have committed the transaction, since it has not voted.

   (b) It crashed after it had voted while it was in its uncertain period.

   Solution;

   Ask the coordinator was happened, whether the transaction was committed or aborted.