

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

**Test Bank: Additional Problems and Solutions**

Michael Kifer, Arthur Bernstein, Philip M. Lewis

# Contents

<b>Problems for Chapter 2: The Big Picture</b>	<b>3</b>
<b>Problems for Chapter 3: The Relational Data Model</b>	<b>6</b>
<b>Problems for Chapter 4: Conceptual Modeling of Databases with E-R and UML</b>	<b>12</b>
<b>Problems for Chapter 5: Relational Algebra and SQL</b>	<b>29</b>
<b>Problems for Chapter 6: Database Design with the Relational Normalization Theory</b>	<b>47</b>
<b>Problems for Chapter 7: Triggers and Active Databases</b>	<b>63</b>
<b>Problems for Chapter 8: Using SQL in an Application</b>	<b>67</b>
<b>Problems for Chapter 9: Physical Data Organization and Indexing</b>	<b>71</b>
<b>Problems for Chapter 10: The Basics of Query Processing</b>	<b>83</b>
<b>Problems for Chapter 11: An Overview of Query Optimization</b>	<b>87</b>
<b>Problems for Chapter 12: Database Tuning</b>	<b>99</b>
<b>Problems for Chapter 13: An Overview of Transaction Processing</b>	<b>104</b>
<b>Problems for Chapter 16: Introduction to Object Databases</b>	<b>106</b>
<b>Problems for Chapter 17: Introduction to XML and Web Data</b>	<b>109</b>

## 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.

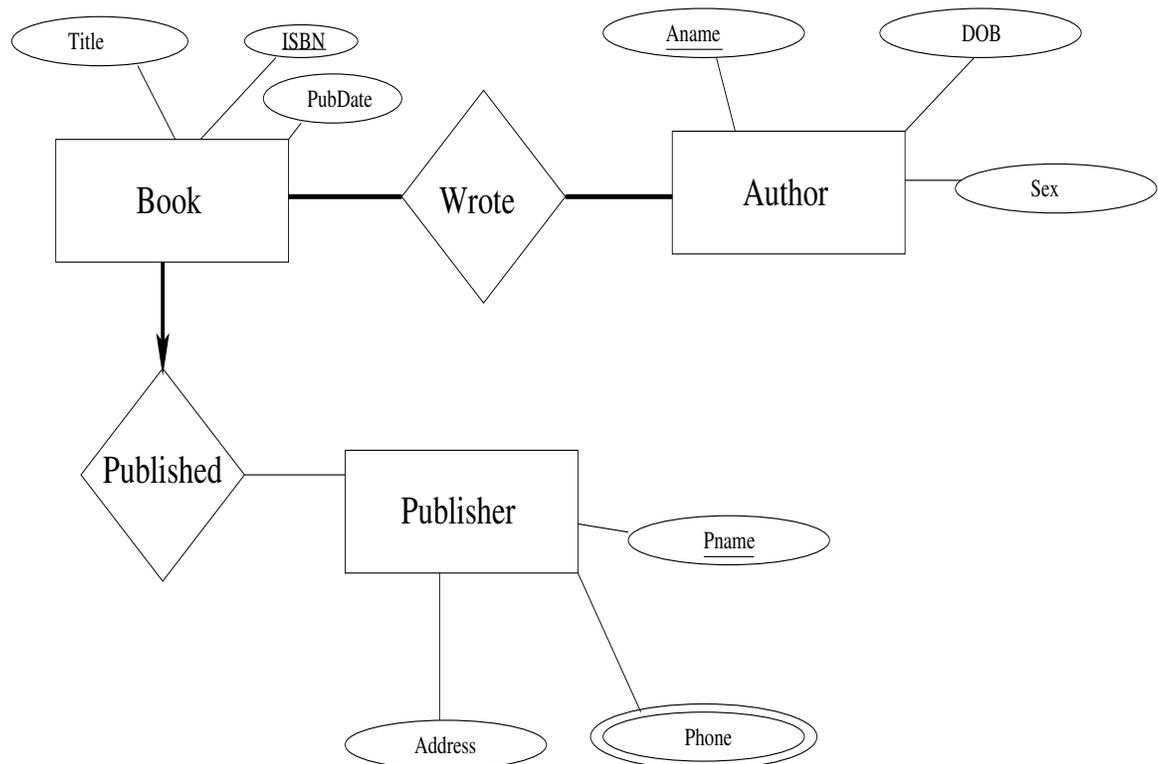
A book can be written by more than one author, but it can be published by only one publisher. Books do not write themselves and do not publish themselves (hint: these are constraints). An author can write more than one book and to be called an author one, of course, has to write at least one book.

- (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.

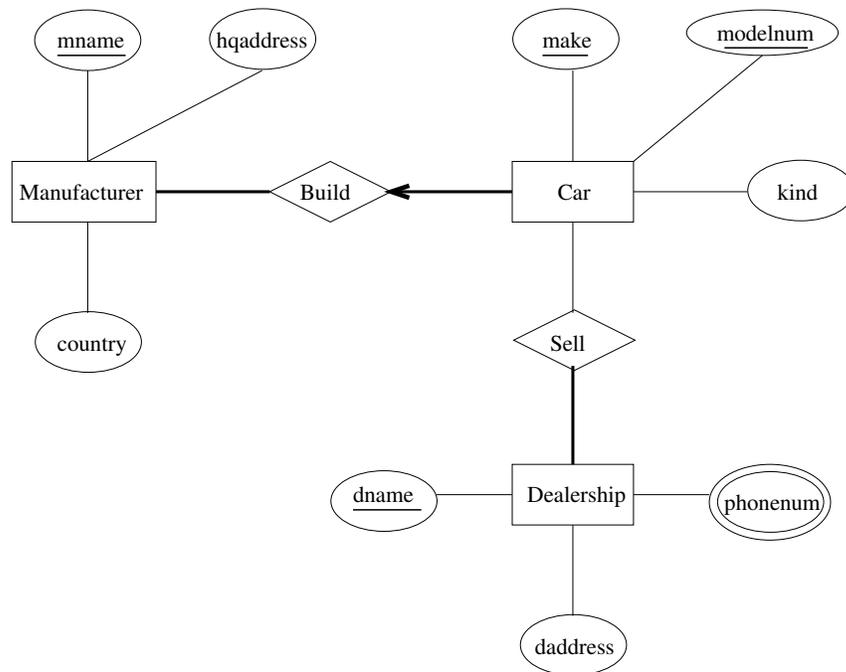
2. Consider the following automotive enterprises, involving cars, car manufacturers, and car dealerships. Car Manufacturers are companies that build cars and they have attributes such as name, headquarters address, country of incorporation, etc. Cars have attributes such as make (e.g., Ford, Volkswagen, 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, convertible, 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 <$100000000000)
```

```
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 HasPHONEUM (
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:

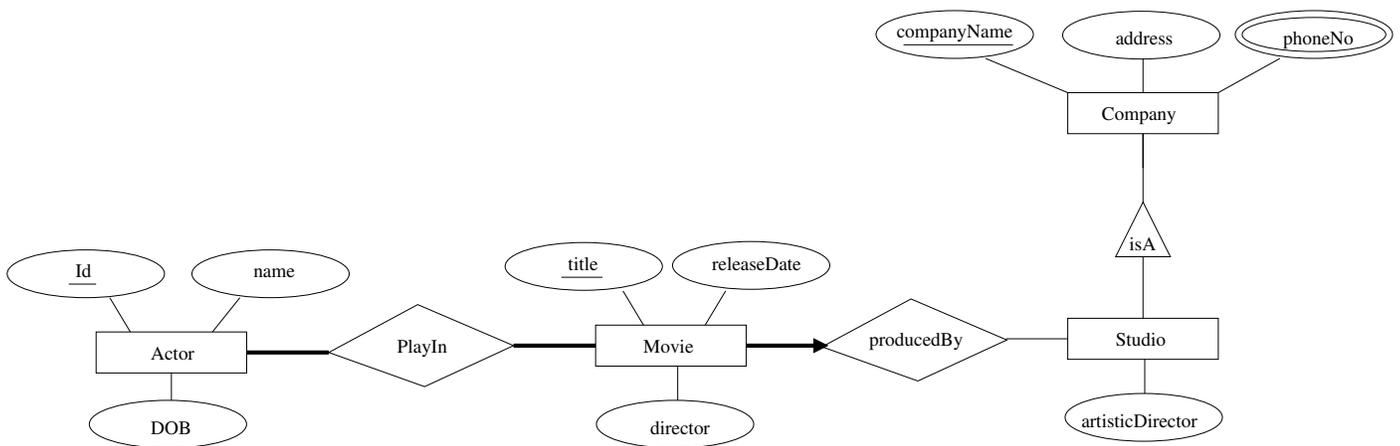
- i. Each manufacturer builds at least one car
- ii. Each dealership sells at least one car

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)
```

```

)

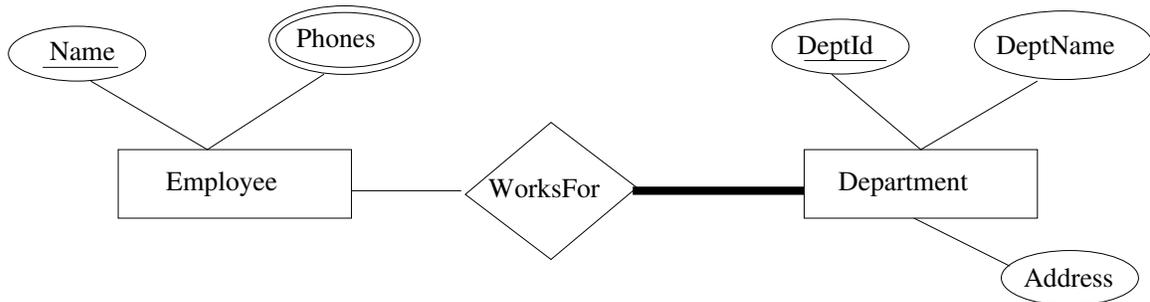
-- Expresses the inclusion dependency that Studio is a Company
CREATE ASSERTION STUDIOIsACOMPANY
CHECK (
    NOT EXISTS ( SELECT *
                  FROM Studio S
                  WHERE S.Name NOT IN
                        ( SELECT C.CompanyName
                          FROM Company C ) )
)

```

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:

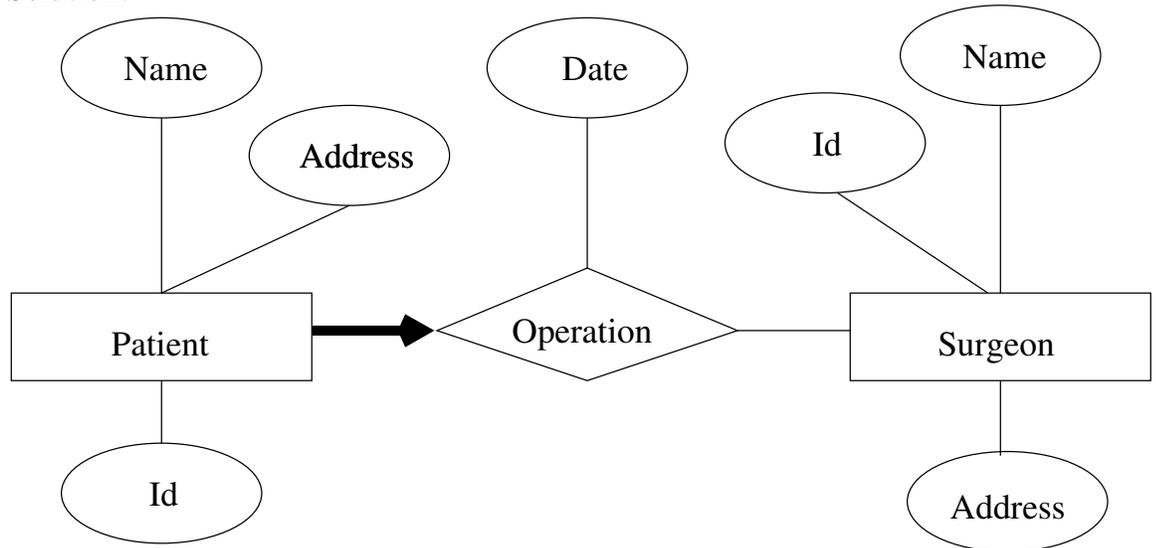
```

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.

Change to schema:

```

CREATE TABLE OPERATION (
    Pid INTEGER,
    Sid INTEGER,
    Date DATE,
    PRIMARY KEY (Sid, Date),
    UNIQUE (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.

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

**Solution:**

```

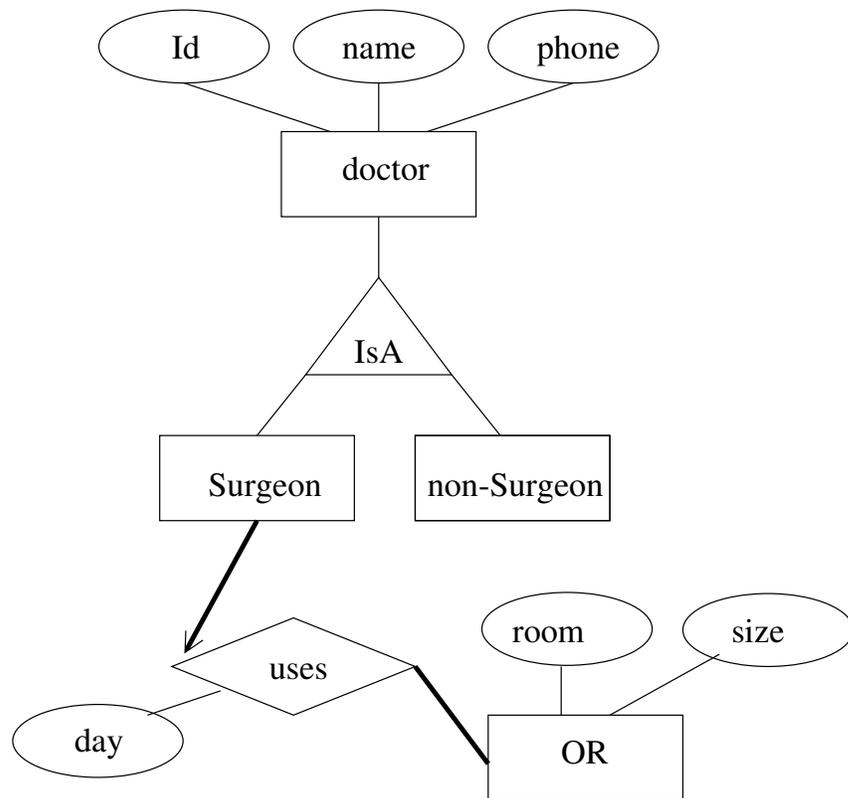
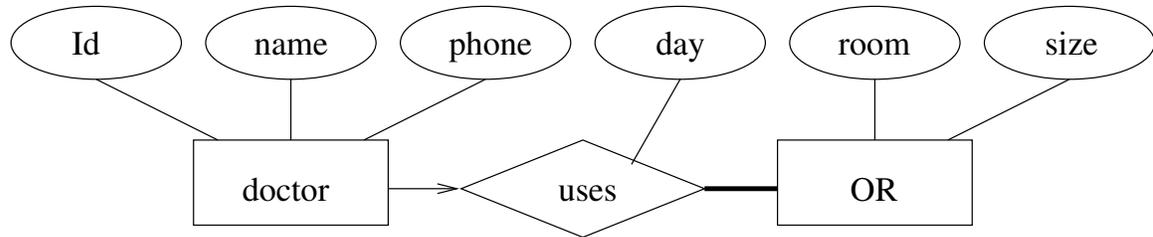
CREATE ASSERTION TOTAL
CHECK NOT EXISTS (SELECT * FROM Surgeons S
WHERE S.Id NOT IN (
SELECT O.Sid FROM OPERATION O))

```

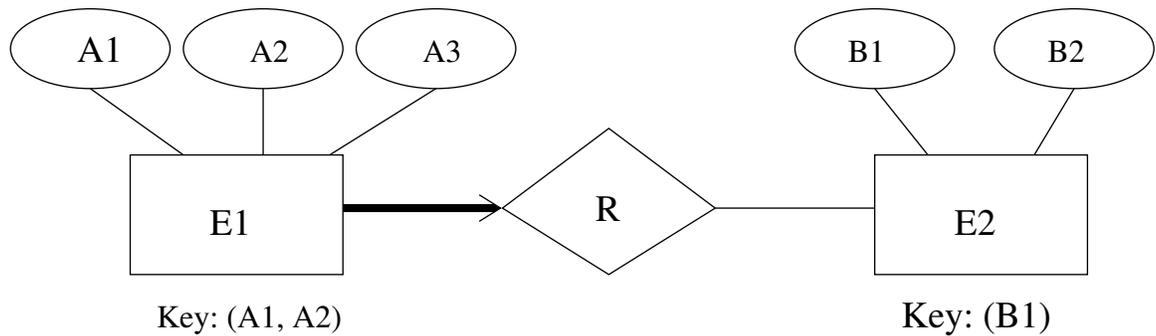
6. In a hospital each surgeon is assigned to a particular operating room (OR) on the same day each week (e.g., Dr. Smith is in OR22 every Wednesday). A surgeon gets exactly one such assignment and each OR gets some use. Other doctors (not surgeons) don't have ORs assigned to them. ORs are described by 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:**

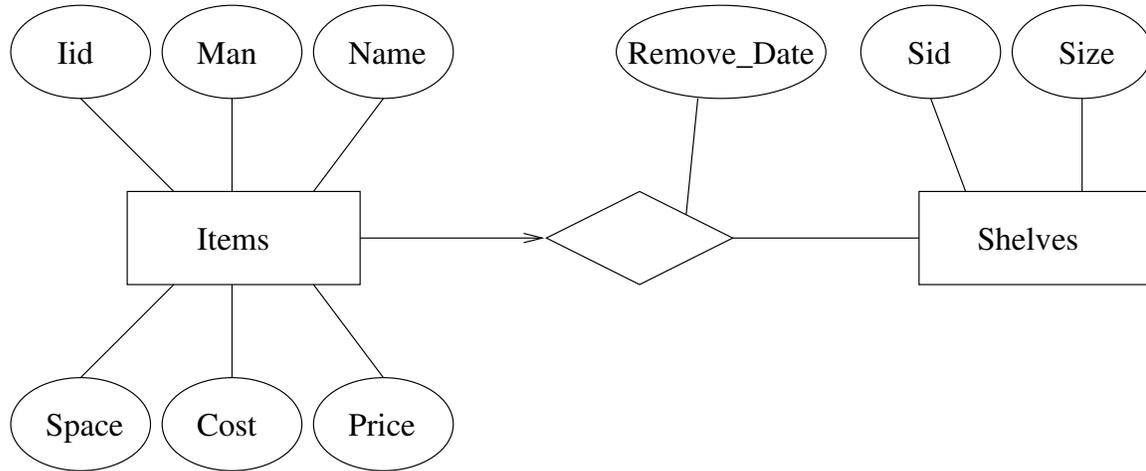
```
CREATE VIEW X (A3, B1) AS
SELECT E1.A3, R.B1
FROM E1, R
WHERE E1.A1 = R.A1 AND E1.A2 = R.A2
```

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

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)  $\pi_{\text{Make}}(\sigma_{\text{MName}=\text{'Mercedes/Chrysler'}}(\text{CAR}))$
- (b)  $\pi_{\text{DName}, \text{DAddress}}(\text{DEALERSHIP} \bowtie \sigma_{\text{Make}=\text{'Plymouth'}}(\text{SELL}))$
- (c)  $\pi_{\text{DAddress}}(((\sigma_{\text{Country}=\text{'Japan'}}(\text{MANUFACTURER}) \bowtie \text{CAR}) \bowtie \text{SELL}) \bowtie \text{DEALERSHIP})$
- (d)  $\pi_{\text{DName}}(\text{DEALERSHIP}) - \pi_{\text{DName}}(\sigma_{\text{MName}=\text{'Mercedes/Chrysler'}}(\text{CAR}) \bowtie \text{SELL})$

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:**

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

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:**

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

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:

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

(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:**

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

(c) Use SQL for the following query:

*Find the travel agents who sold more than 5 tickets.*

**Solution:**

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

(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.ProfId 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.

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

**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 '11111111'.

**Solution:**

```
SELECT T.CrsCode
FROM TRANSCRIPT T
WHERE T.CrsCode LIKE 'CS%' AND T.StudId = '11111111'
```

- (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:  $AB \rightarrow CD$ ,  $BC \rightarrow FG$ ,  $A \rightarrow G$ ,  $G \rightarrow B$ ,  $C \rightarrow G$ .
  - (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

$AB \rightarrow C$   
 $AB \rightarrow D$   
 $BC \rightarrow F$   
 $BC \rightarrow G$   
 $A \rightarrow G$   
 $G \rightarrow B$   
 $C \rightarrow G$

#### Step 2 - Reduce the left-hand sides

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

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

#### Step 3 - eliminate redundant FDs

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

$A \rightarrow C$   
 $A \rightarrow D$   
 $C \rightarrow F$   
 $G \rightarrow B$   
 $C \rightarrow G$

- (b) The decomposition into ABCD and CFG is lossless, since the intersection is C and  $C^+ = CFGB$ . In particular, the FDs imply  $C \rightarrow CFG$ , 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.  $ABE \rightarrow CK$
2.  $AB \rightarrow D$
3.  $C \rightarrow BE$
4.  $EG \rightarrow DHK$
5.  $D \rightarrow L$
6.  $DL \rightarrow EK$
7.  $KL \rightarrow DM$

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

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

*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.  $AB \rightarrow C$
3.  $AB \rightarrow D$
4.  $C \rightarrow B$
5.  $C \rightarrow E$
6.  $EG \rightarrow D$
7.  $EG \rightarrow H$
9.  $D \rightarrow L$

10.  $D \rightarrow E$
11.  $D \rightarrow K$
12.  $KL \rightarrow D$
13.  $KL \rightarrow M$

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

**Solution:**

$\mathbf{R}_1 = (\text{ABCD}, \{\text{AB} \rightarrow \text{CD}\})$

$\mathbf{R}_2 = (\text{CBE}, \{\text{C} \rightarrow \text{BE}\})$

$\mathbf{R}_3 = (\text{EGDH}, \{\text{EG} \rightarrow \text{DH}\})$

$\mathbf{R}_4 = (\text{DLEK}, \{\text{D} \rightarrow \text{LEK}\})$

$\mathbf{R}_5 = (\text{KLDM}, \{\text{KL} \rightarrow \text{DM}\})$

- (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  $\text{C} \rightarrow \text{B}$ : Split into CB and CDA.

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

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

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

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

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

and the following multivalued dependencies:

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

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

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

So 3NF is

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

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

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

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:

$ABF \rightarrow C$

$CF \rightarrow B$

$CD \rightarrow A$

$BD \rightarrow AE$

$C \rightarrow F$

$B \rightarrow F$

(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

ABDEF

ABDEFC

by  $BD \rightarrow E$

by  $B \rightarrow F$

by  $ABF \rightarrow C$

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

**Solution:**

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

Step 2:  $ABF \rightarrow C$  can be replaced with:  $AB \rightarrow C$  while  $CF \rightarrow B$  can be replaced with:  $C \rightarrow B$ . This is because  $AB \rightarrow C$  is entailed by  $ABF \rightarrow C$  and  $C \rightarrow F$ , and  $C \rightarrow B$  is entailed by  $CF \rightarrow B$  and  $C \rightarrow F$ .

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

$AB \rightarrow C$   
 $C \rightarrow B$   
 $CD \rightarrow A$   
 $BD \rightarrow A$   
 $BD \rightarrow E$   
 $C \rightarrow F$   
 $B \rightarrow F$

Step 3: Eliminate redundant FDs:

$C \rightarrow F$  is entailed by  $C \rightarrow B$  and  $B \rightarrow F$ , while

$CD \rightarrow A$  is entailed by  $C \rightarrow B$  and  $BD \rightarrow A$ . Therefore, the minimal cover is

$AB \rightarrow C$   
 $C \rightarrow B$   
 $BD \rightarrow A$   
 $BD \rightarrow E$   
 $B \rightarrow F$

- (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

$$\mathbf{R}_1 = (ABC, \{AB \rightarrow C\})$$

$$\mathbf{R}_2 = (CB, \{C \rightarrow B\})$$

$$\mathbf{R}_3 = (BDAE, \{BD \rightarrow A, BD \rightarrow E\})$$

$$\mathbf{R}_4 = (BF, \{B \rightarrow F\})$$

Test for losslessness:

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

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

$$\mathbf{R}_{12} = (CA, \{\})$$

This decomposition is not dependency preserving, since the dependency  $AB \rightarrow C$  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<sup>rd</sup> 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:

$A T C \rightarrow B \text{ Sh Sub}$   
 $A \rightarrow B$   
 $\text{Sh} \rightarrow \text{Sub}$

The relation can be decomposed into the following three relations:

(A T C Sh) with FD  $ATC \rightarrow \text{Sh}$   
(A B) with FD  $A \rightarrow B$   
(Sh Sub) with FD  $\text{Sh} \rightarrow \text{Sub}$

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  $X \rightarrow A$  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  $D \rightarrow C$  and  $AB \rightarrow CDE$ . 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  $D \rightarrow C$  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  $AB \rightarrow CDE$ ,  $D \rightarrow E$ , and  $CD \rightarrow B$ .

(a) What is a key of R?

**Solution:**

AB

(b) What does the FD  $D \rightarrow E$  mean?

**Solution:**

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

(c) What does the FD  $D \rightarrow E$  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  $D \rightarrow E$  we might try one of the following.

i. We might add the constraint UNIQUE (D,E). Would this be an appropriate way 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 3<sup>rd</sup> Normal Form? Explain.

**Solution:**

No. The FD  $D \rightarrow E$  violates the conditions for 3<sup>rd</sup> 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  $R1 \cap R2 \rightarrow R1$  or  $R1 \cap R2 \rightarrow R2$

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

**Solution:**

R1: ABCD with FDs  $AB \rightarrow CD$ ,  $CD \rightarrow B$

R2: DE with FD  $D \rightarrow E$

R1 is in 3<sup>rd</sup> Normal Form since AB is a key and B is part of a key.

R2 is in 3<sup>rd</sup> Normal Form since D is a key.

lossless:  $\{ABCD\} \cap \{DE\} = \{D\}$  and D is a key of R2.

7. Is R1 in BCNF? Is R2 in BCNF? Explain.

**Solution:**

R1 is not in BCNF because of FD  $CD \rightarrow B$  since CD is not a superkey.

R2 is in BCNF since D is a key.

8. (a) If the functional dependency  $X \rightarrow Y$  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  $X \subseteq Y$ . 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  $X \rightarrow Y$ . Since  $k$  contains all attributes, it contains  $X$  and  $Y$ . But the FD says that  $X$  determines  $Y$ . Hence we can delete  $Y - (X \cap Y)$  from  $k$  and the remaining attributes still form a key. This means that  $k$  is not minimal, a contradiction.

9. 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.  $R_1 = (ABDEFGH, \{BE \rightarrow GH, G \rightarrow FA, F \rightarrow B\})$

$R_2 = (DC, \{D \rightarrow C\})$

$R_2$  is in BCNF since D is a key.  $R_1$  is not in BCNF since the LHS of all FDs are not keys.

ii.  $R_3 = (BCDEGH, \{BE \rightarrow GH, D \rightarrow C\})$

$R_4 = (GFA, \{G \rightarrow FA\})$

$R_4$  is in BCNF since G is a key.  $R_3$  is not in BCNF since neither BE or D is a key.

iii.  $R_5 = (ACDEFGH, \{G \rightarrow FA, D \rightarrow C\})$

$R_6 = (FB, \{F \rightarrow B\})$

$R_6$  is in BCNF since F is a key.  $R_5$  is not in BCNF since neither D or G is a key.

iv.  $R_7 = (ABCDEF, \{F \rightarrow B, D \rightarrow C\})$

$R_8 = (BEGH, \{BE \rightarrow GH\})$

$R_8$  is in BCNF since BE is a key.  $R_7$  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  $L^+ = (L1 \cup L2)^+$ . In the first decomposition  $L = L1 \cup L2$  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  $\{F \rightarrow B, D \rightarrow C, BE \rightarrow GH\}$  does not contain  $FA$  (it is just  $G$ ).

- (h) \* Consider a schema  $(\bar{R}, \mathcal{F})$ , where  $\bar{R}$  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  $\bar{X} \rightarrow \bar{Y}$ , where the left-hand side,  $\bar{X}$ , 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,  $A_0 \in \bar{R}$ , 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  $\bar{X} \rightarrow C \in \mathcal{F}^+$  such that  $\bar{X}$  is not a superkey,  $C \in \bar{K}$ , for some key  $\bar{K}$ .

Since  $\bar{K}$  is a key, there must be an FD of the form  $\bar{Y} \rightarrow A_0 \in \mathcal{F}$ , where  $\bar{Y} \subseteq \bar{K}$  (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  $\bar{Y}$  is a singleton attribute. Furthermore,  $\bar{Y}$  must be equal  $\bar{K}$  or else  $\bar{Y} \rightarrow A_0$  violates 3NF (since  $A_0$  does not belong to any key). Therefore,  $\bar{K}$  is a set that contains only one attribute. Since  $C \in \bar{K}$ , it follows that  $\bar{K} = \{C\}$ , *i.e.*,  $\{C\}$  is a key. But then, since  $\bar{X} \rightarrow C \in \mathcal{F}^+$ , it follows that  $\bar{X}$  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.Quantity
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 O
FOR EACH ROW
WHEN ( EXISTS ( SELECT S.Id
                FROM STUDENT S, TRANSCRIPT T
                WHERE S.Status = 'good' AND T.StudId = O.Id
                AND S.Id = O.Id
                GROUP BY S.Id
                HAVING avg(T.Grade) =< 3 ) )
UPDATE STUDENT
SET Status = 'bad'
WHERE Id = O.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 than 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

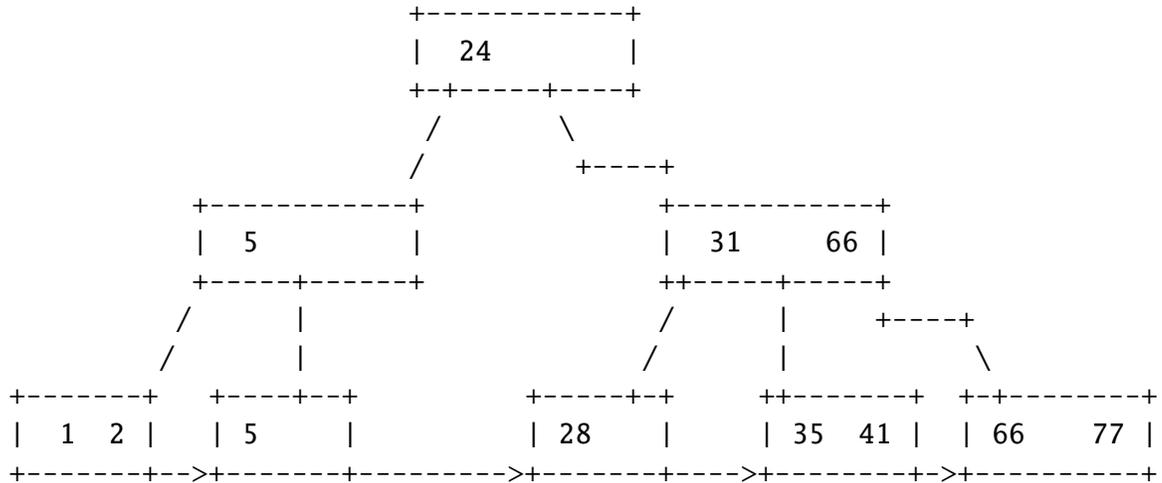
## Problems for Chapter 9: Physical Data Organization and Indexing

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:**

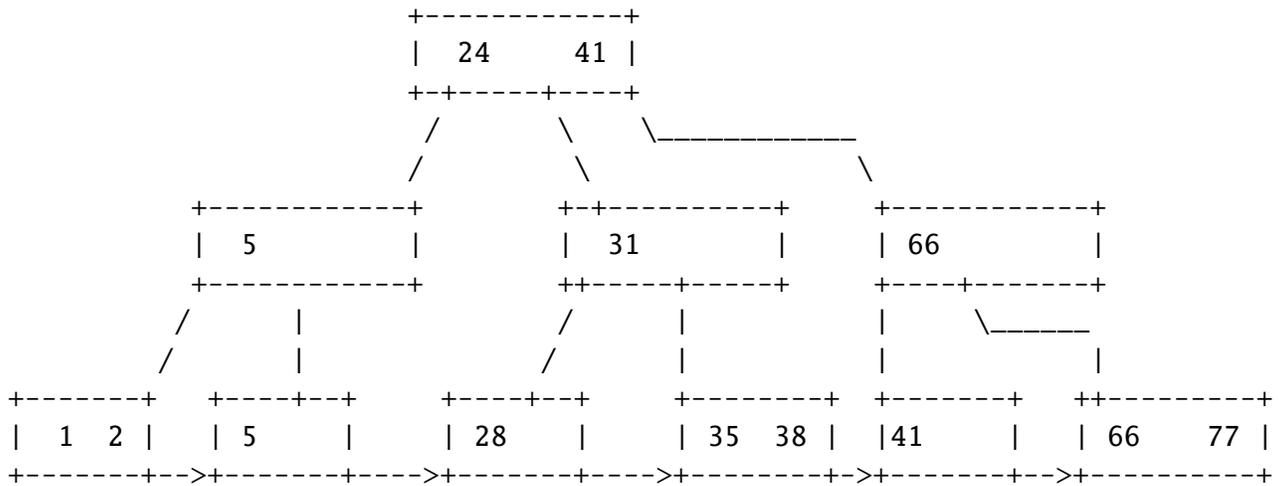
Hash each relation on all attributes using the same hash function. Keep the distinction between the two parts of the bucket, which came from different relations. Tuples that fall into the intersection must fall into the same bucket. If the bucket is small enough, do the intersection in main memory. Else 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  $\text{ceiling}(\log_{100}5000) = 2$ .



5. Assume you have a table that contains 27,000 records, and you have an *unclustered* B<sup>+</sup>-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<sup>+</sup>-tree index for this table? Draw a sketch of this "maximal height" B<sup>+</sup>-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 if 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  $1 + \text{ceiling}(\log_{50} 13500) = 4$

- (b) What is the **minimum** possible number of levels in the B<sup>+</sup>-tree index for this table? Draw a sketch of this "minimal height" B<sup>+</sup>-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  $\text{ceiling}(\log_{100} 27000) = 3$ .

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<sup>+</sup> 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*?

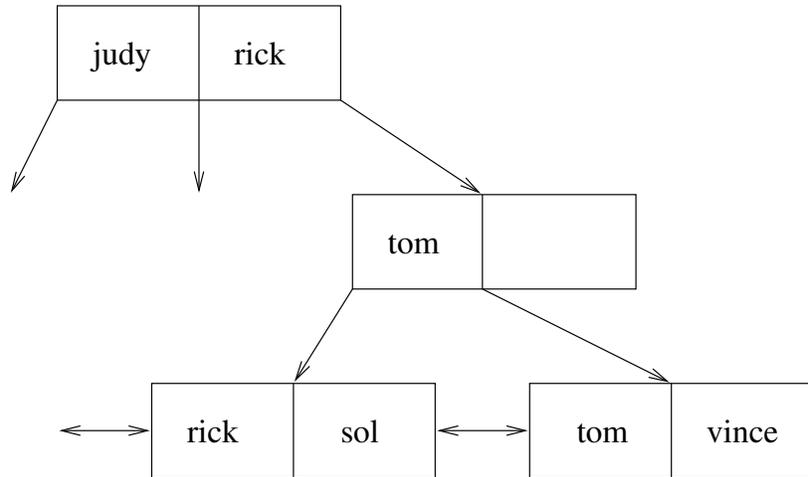


Figure 1: Initial state of B<sup>+</sup> tree.

**Solution:**

See Figure 2.

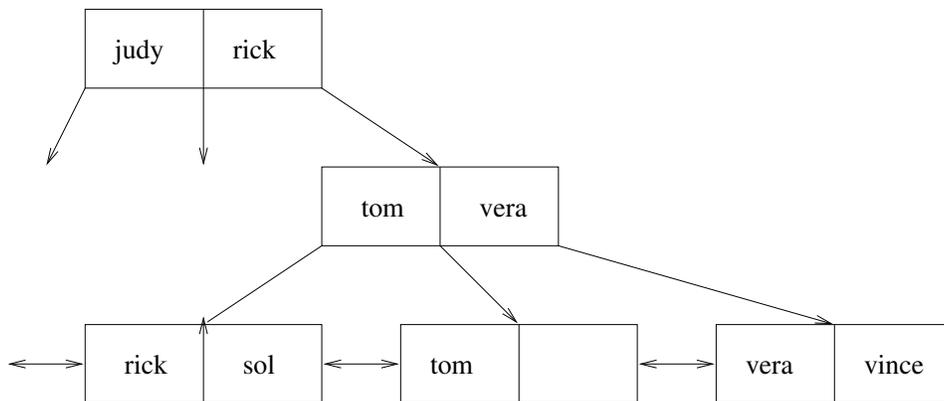


Figure 2: State of B<sup>+</sup> tree after the insertion of *vera*.

(b) What does the tree look like if *rob* is inserted (after *vera* has been inserted)?

**Solution:** See Figure 3.

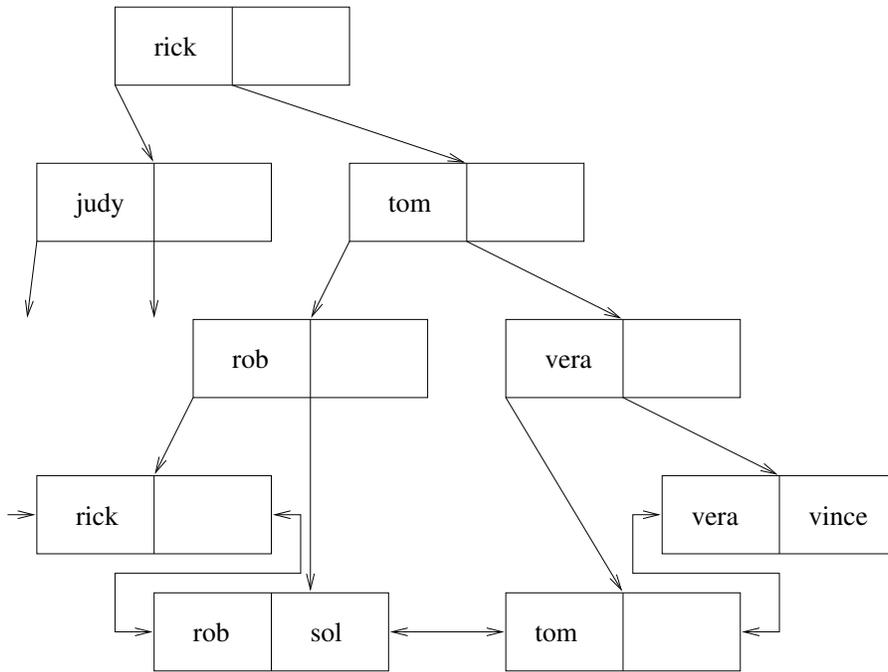


Figure 3: State of B<sup>+</sup>-tree after the insertion of *rob*.

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 =  $2500 * 20\text{ms} = 50$  sec

(b) An unclustered B<sup>+</sup> 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

$4 * 20\text{ms}$  per I/O operation = 80 ms

(c) A clustered B<sup>+</sup> 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)

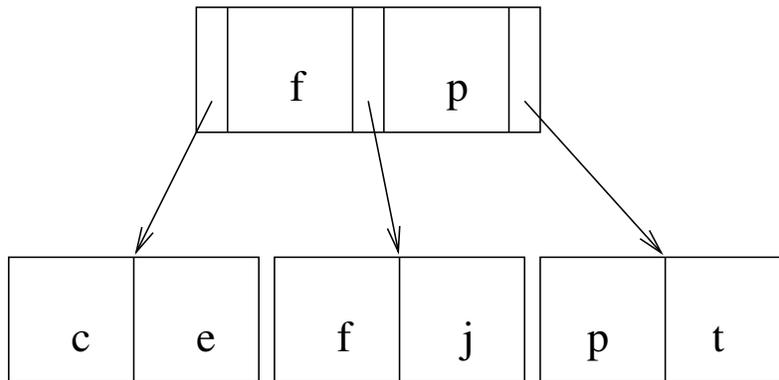
number of separator levels in tree is smallest integer  $k$  satisfying  $100^k \geq 5000$

hence  $k = 2$

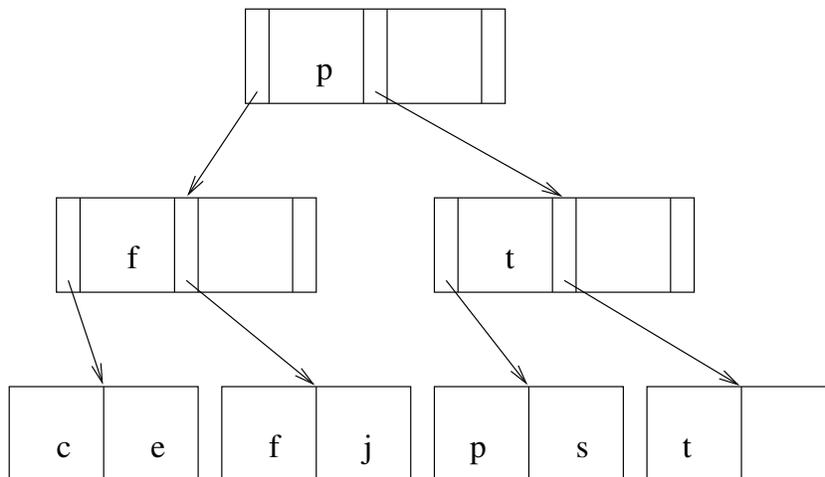
total number I/O operations = 2 (to search separator levels) + 1 (to get index page)

cost = 60 ms

10. Search key value  $s$  is to be inserted in the following B<sup>+</sup> 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,  $A_1, A_2, \dots, A_n$ . 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  $A_1$  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 =  $(n * r * R)/P = \text{cost}$

- (b) There is a non-integrated (separate index file) clustered index with search key  $A_1$ .

**Solution:**

Search down the tree using  $A_1 = 25$ , follow pointer to first qualifying row in data file, scan data file and return all rows until row with  $A_1 > 50$  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 =  $\log_{P/r} R = \text{cost of tree search}$

cost of file scan =  $.25((n * r * R)/P)$  since approximately a quarter of the rows have to be returned (values of  $A_1$  are randomly distributed)

total cost =  $\log_{P/r} R + 1 + .25((n * r * R)/P)$

- (c) There is an unclustered index with search key  $(A_2, A_1)$ .

**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  $r \bowtie_{A < B} s$  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  $\sigma_{A>1000 \wedge A<6000}(s)$ .

(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  $5000/100000 = 1/20$  of 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 ( $\text{ceiling}(250/100)=3$ ). 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

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

and estimate the cost of using this access path.

**Solution:**

Since we have a disjunction, we need to evaluate  $\sigma_{\text{Status}='U3'}(\text{STUDENT})$  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  $r \bowtie_{A=B} s$ . 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:**

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

- (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  $\sigma_{\text{DeptName}='Sales'}(\text{DEPARTMENT})$  first. This is supposed to produce a single tuple. Selection  $\sigma_{\text{Building}='E213'}$  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:**

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

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:**

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

- (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:**

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

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  $PID \rightarrow Name, Budget$  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<sup>+</sup> tree

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

**Solution:**

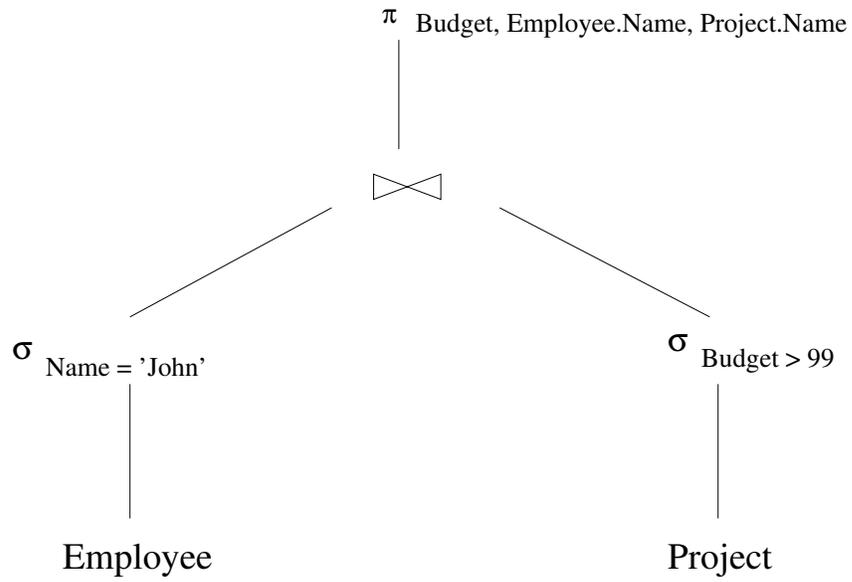
There are three possible candidates to consider: (1) push both selections; (2) push selection to EMPLOYEE only; (3) push selection to PROJECT only. Option (1) yields the least number of I/Os.

Selection  $\sigma_{Name='John'}(EMPLOYEE)$ : 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  $\sigma_{Budget>99}(PROJECT)$ : 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<sup>+</sup> tree, this will take  $2 + \text{“the number of result pages”}$  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  $\rightarrow$  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<sup>+</sup> tree

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

**Solution:**

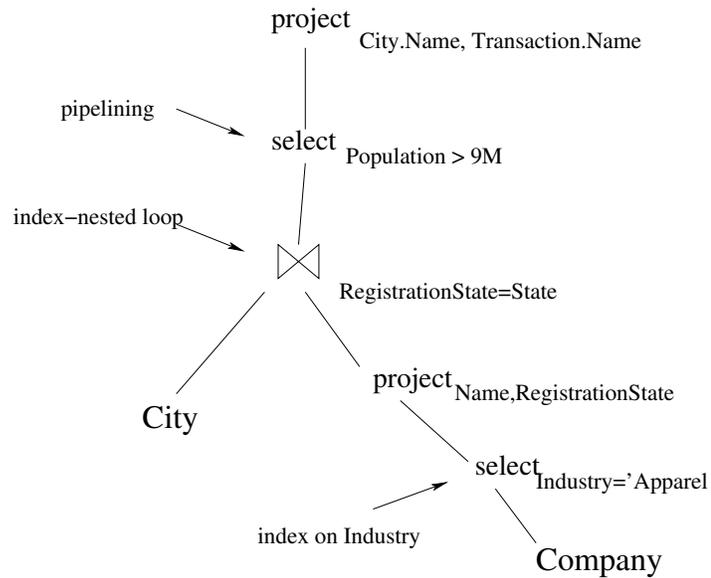
The index on COMPANY cannot be used in a join, so it makes sense to push the selection and the projection to COMPANY. The size of  $\pi_{\text{Name,RegistrationState}}(\sigma_{\text{Industry=Apparel}}(\text{COMPANY}))$  is:  $20000/50 = 400$  tuples (after selection) = 8 pages. Projection further reduces 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 Crry is clustered on State, this relation is ordered on the State attribute. Therefore, we can pull  $\pi_{Name,RegistrationState}(\sigma_{Industry=Apparel}(COMPANY))$  through the 6-page buffer using just one page and join it with Crry 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)

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

Consider the following query:

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

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.

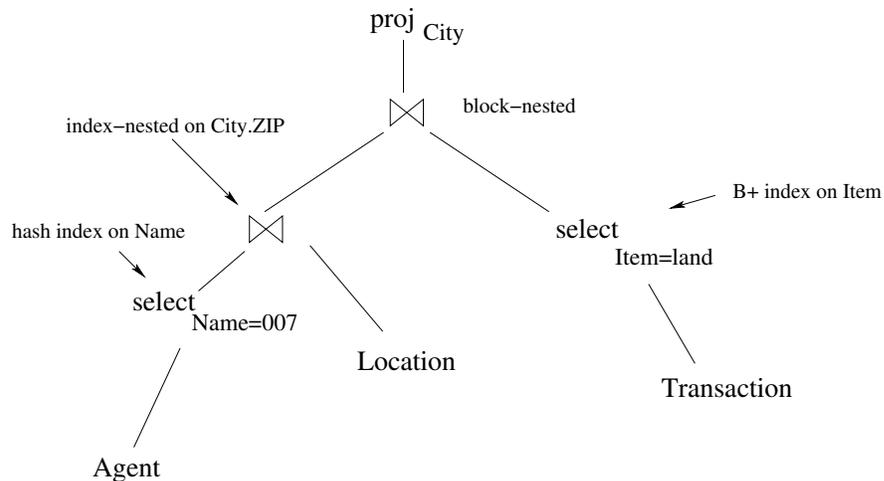
Assume 50 buffer pages and the following statistics and indices:

- AGENT: 1,000 tuples, 5 tuples/page; 200 agents.  
 Clustered hash index on Name.  
 Unclustered B<sup>+</sup> 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<sup>+</sup> 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:

- $\sigma_{\text{Name}='007'}(\text{AGENT})$ : 5 tuples, 1 page, 1 I/O (since Name is clustered).
- $\sigma_{\text{Item}='land'}(\text{TRANSACTION})$ : 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  $50+20=70$ pages.

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

7. (a) The equivalence

$$\pi_a(R \bowtie_c S) \equiv \pi_{a_1}(R) \bowtie_c \pi_{a_2}(S)$$

where  $a_1$  and  $a_2$  are disjoint subsets of  $a$  satisfying  $a = a_1 \cup a_2$  holds under certain conditions on  $a_1$ ,  $a_2$ , and  $c$ . State the conditions.

**Solution:**

$a_1$  and  $a_2$  are subsets of  $a$  corresponding to  $R$  and  $S$  and all attributes named in  $c$  are contained in  $a_1$  and  $a_2$

- (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:**

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

- (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:

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

or

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

## 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<sup>+</sup> tree/hash).

**Solution:**

B<sup>+</sup> tree with search key (DeptId, Name), unclustered

- (b) If your answer to the previous part was a hash (B<sup>+</sup> tree) explain whether a B<sup>+</sup> 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      (2) SELECT AVG (Salary)
FROM EMPLOYEE E              FROM EMPLOYEE E
WHERE E.Name = :name          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<sup>+</sup> tree or hash, clustered or unclustered.

**Solution:**

clustered B<sup>+</sup> tree (or hash) on DeptId

- ii. Describe all additional indexes (don't use index covering). For each state whether it is a B<sup>+</sup> tree or hash, clustered or unclustered.

**Solution:**

unclustered on primary key Id, could be B<sup>+</sup> tree or hash

unclustered on Name, could be B<sup>+</sup> 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. The SELECT statement

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

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

- (a) P is (B = 10)

**Solution:**

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

- (b) P is (B > 10)

**Solution:**

clustered B<sup>+</sup> tree on B, unclustered on K  
rows are ordered on B, search for B=10 and scan from that point

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

**Solution:**

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

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

**Solution:**

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

3. 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<sup>+</sup> tree, hash) is used.

**Solution:**

clustered or unclustered, hash or B<sup>+</sup> 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<sup>+</sup> tree, hash) is used for each statement. Explain your reasoning.

**Solution:**

Use an unclustered B<sup>+</sup> 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<sup>+</sup> 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<sup>+</sup> 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.

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

## Problems for Chapter 13: 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 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

$$r_1(x) \ w_2(x) \ w_2(y) \ c_2 \ r_2(y) \ c_1$$

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 its 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 what happened, whether the transaction was committed or aborted.

## Problems for Chapter 16: Introduction to 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)
```

## Problems for Chapter 17: Introduction to 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>

```

```

    </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>
```

```

        </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>

```

```

    </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>

```