Homework 3

Ruili Yao

2021146833

Chapter 6

6.16. Specify the following queries on the COMPANYrelational database schema shown in Figure 3.5, using the relational operators discussed in this chapter. Also show the result of each query as it would apply to the database state in Figure 3.6.

a. Retrieve the names of all employees in department 5 who work more than

10 hours per week on the ProductX project.

b. List the names of all employees who have a dependent with the same first name as themselves.

d. For each project, list the project name and the total hours per week (by all employees) spent on that project.

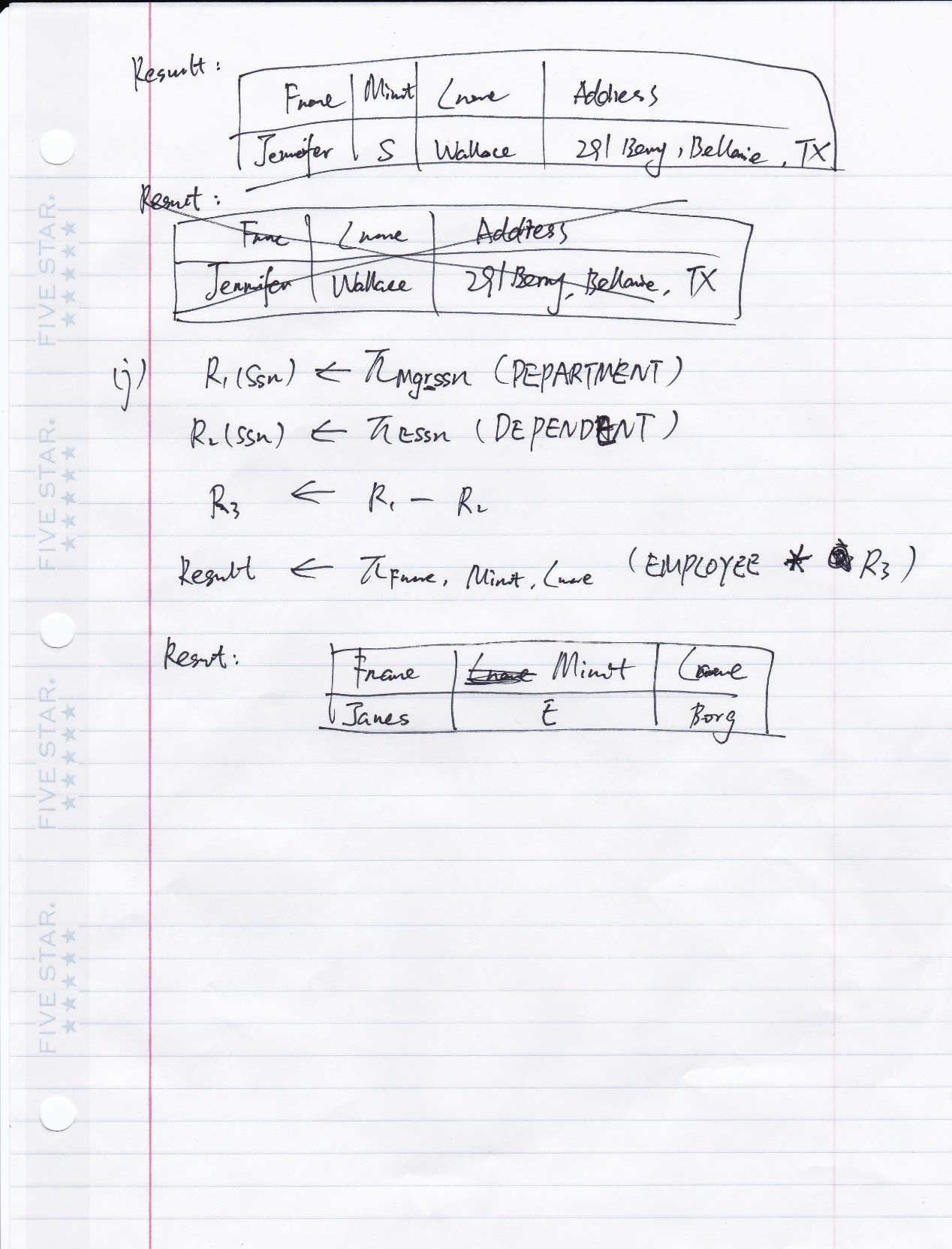
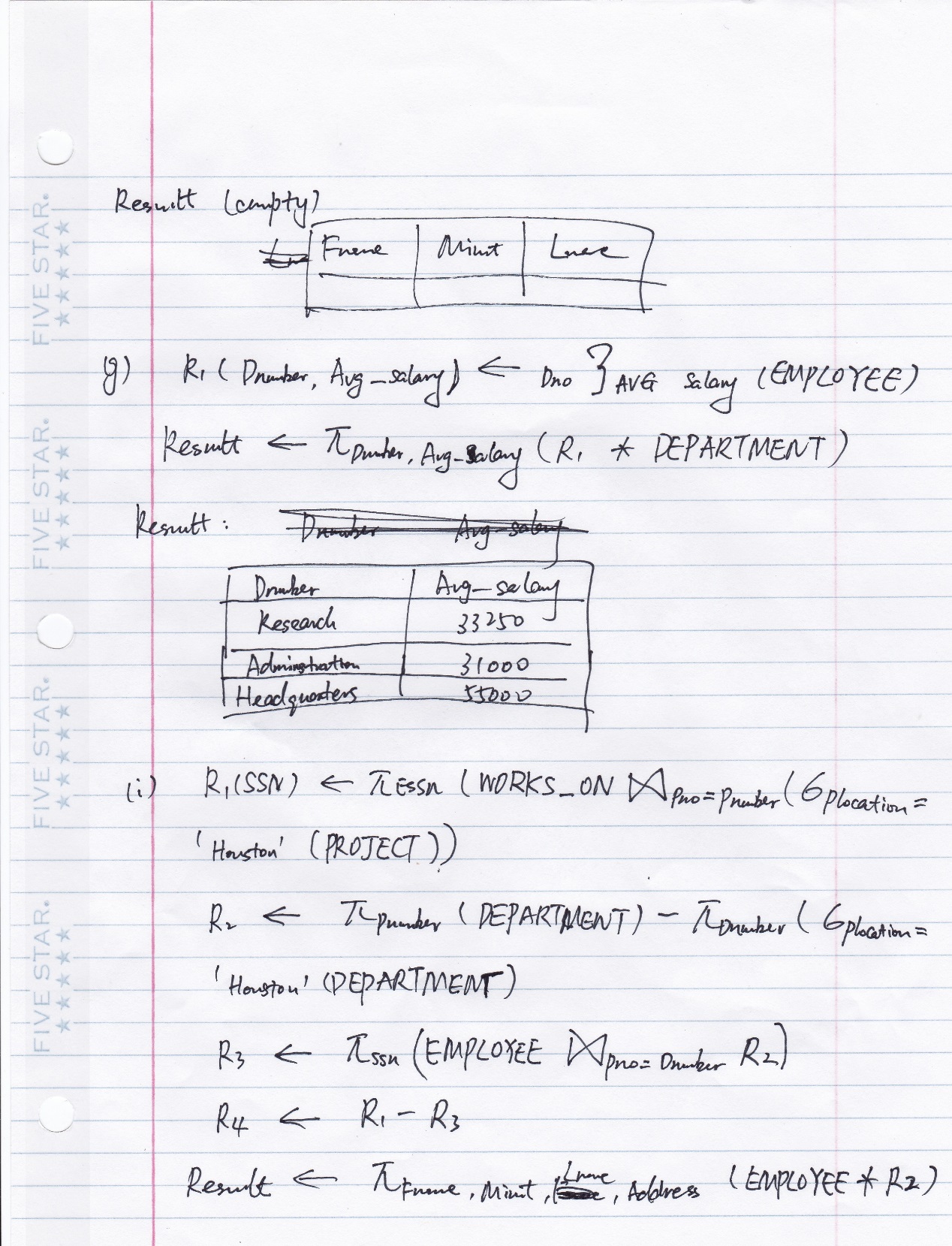
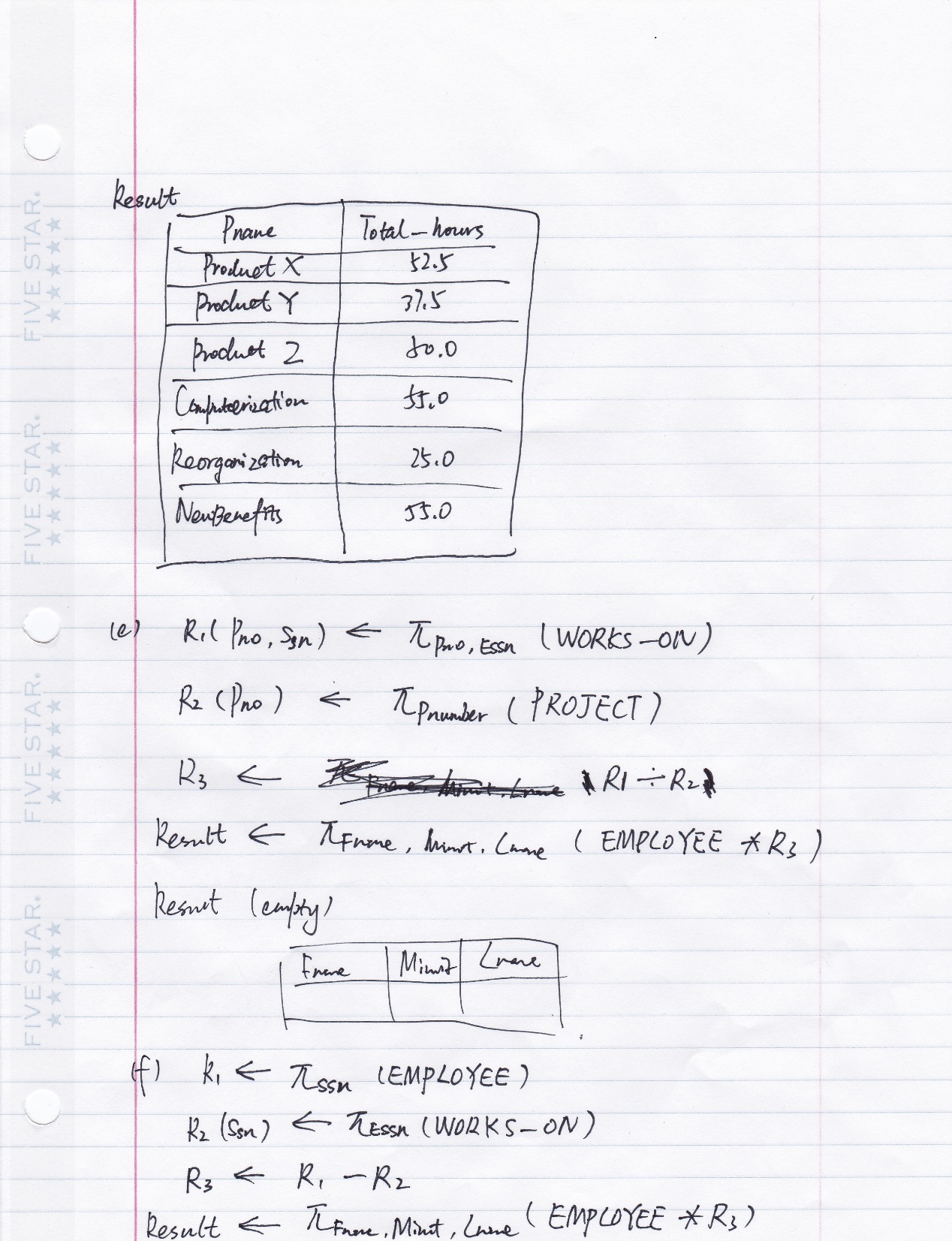
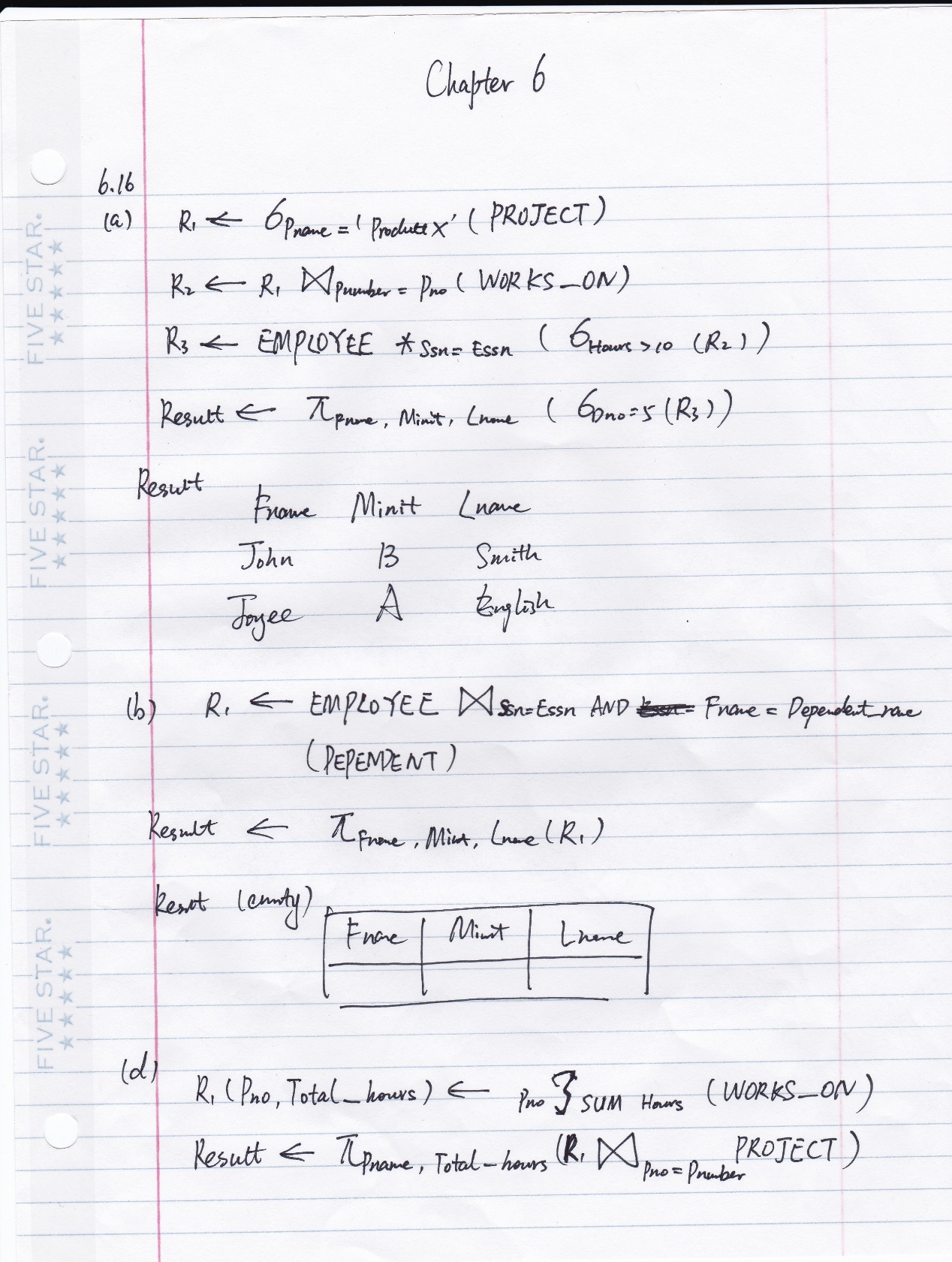
e. Retrieve the names of all employees who work on every project.

f. Retrieve the names of all employees who do not work on any project.

g. For each department, retrieve the department name and the average salary of all employees working in that department.

i. Find the names and addresses of all employees who work on at least one project located in Houston but whose department has no location in Houston.

j. List the last names of all department managers who have no dependents.



Chapter 4

1. Retrieve the names of employees in department 5 who work more than 10 hours per week on the 'ProductX' project.

SELECT Fname. Minit, Lname

FROM EMPLOYEE, WORKS\_ON, PROJECT

WHERE Dno=5 AND Ssn= Essn AND Pno=Pnumber AND Pname = ‘ProductX’ AND Hours>10

#################################

SELECT LNAME, FNAME

FROM EMPLOYEE

WHERE DNO=5 AND SSN IN ( SELECT ESSN

FROM WORKS\_ON

WHERE HOURS>10 AND PNO IN ( SELECT PNUMBER

FROM PROJECT

WHERE PNAME='ProductX' ) ); ####################################

(b) List the names of employees who have a dependent with the same first name as themselves.

SELECT Fname, Minit, Lname

FROM EMPLOYEE, DEPENDENT

WHERE Ssn=Essn AND Fname=Dependent\_name;

###################################

Another possible SQL query uses nesting as follows:

SELECT LNAME, FNAME

FROM EMPLOYEE

WHERE EXISTS ( SELECT \*

FROM DEPENDENT

WHERE FNAME=DEPENDENT\_NAME

AND SSN=ESSN ); ########################################

(d) For each project, list the project name and the total hours per week (by all employees) spent on that project.

SELECT Pname, SUM (Hours)

FROM PROJECT, WORKS\_ON

WHERE Pnumber=Pno

GROUP BY Pname;

(e) Retrieve the names of employees who work on every project. \

SELECT Fname, Minit, Lname

FROM EMPLOYEE

WHERE NOT EXISTS

( SELECT Pnumber

FROM PROJECT

WHERE NOT EXISTS ( SELECT \*

FROM WORKS\_ON

WHERE Pnumber=Pno AND Essn=Ssn ) );

(f) Retrieve the names of employees who do not work on any project.

SELECT Fname, Minit, Lname

FROM EMPLOYEE

WHERE NOT EXISTS

( SELECT \*

FROM WORKS\_ON

WHERE ESSN=SSN )

(g) For each department, retrieve the department name, and the average salary of employees working in that department.

SELECT Dname, AVG (Salary)

FROM DEPARTMENT, EMPLOYEE

WHERE Dnumber=Dno

GROUP BY Dname

1. Find the names and addresses of employees who work on at least one project located in Houston but whose department has no location in Houston.

SELECT Fname, Minit, Lname, Address

FROM EMPLOYEE

WHERE EXISTS

( SELECT \*

FROM WORKS\_ON, PROJECT

WHERE Ssn=Essn AND Pno=Pnumber AND Plocation='Houston' )

AND NOT EXISTS

( SELECT \*

FROM DEPT\_LOCATIONS

WHERE Dno=Dnumber AND Dlocation='Houston' )

(j) List the last names of department managers who have no dependents.

SELECT Fname, Minit, Lname

FROM EMPLOYEE

WHERE EXISTS

( SELECT \*

FROM DEPARTMENT

WHERE Ssn=Mgr\_ssn )

AND NOT EXISTS

( SELECT \*

FROM DEPENDENT

WHERE Ssn=Essn )

4.12. Specify the following queries in SQL on the database schema of Figure 1.2.

a. Retrieve the names of all senior students majoring in ‘CS’ (computer science).

SELECT Name

FROM STUDENT

WHERE Major = ‘CS’ (AND Class = 4)

b. Retrieve the names of all courses taught by Professor King in 2007 and 2008.

SELECT C.Course\_name

FROM COURSE AS C, SECTION AS S

WHERE C.Course\_number = S. Course\_number AND Instructor = ‘King’ AND (Year= 07 OR Year = 08)

c. For each section taught by Professor King, retrieve the course number, semester, year, and number of students who took the section.

SELECT Course\_number, Semester, Year, COUNT(G.Student\_number) AS ‘Number of Students’

FROM SECTION AS S, GRADE\_REPORT AS G

WHERE S.Section\_identifier = G.Section\_identifier AND Instructor = ‘King’

GROUP BY Course\_number, Semester, Year

d. Retrieve the name and transcript of each senior student (Class = 4) majoring in CS. A transcript includes course name, course number, credit hours, semester, year, and grade for each course completed by the student.

SELECT Name, Course\_name, C.Course\_number, Credit\_hours, Semester, Year, Grade

FROM STUDENT AS ST, GRADE\_REPORT AS G, COURSE AS C, SEMESTER AS S

WHERE ST.Student\_number = G.Student\_number AND G.Section\_identifier = S. Section\_identifier AND S.Course\_number = C.Course\_number AND Class = 4 AND Major = ‘CS’

4.13. Write SQL update statements to do the following on the database schema shown in Figure 1.2.

b. Change the class of student ‘Smith’ to 2.

UPDATE STUDENT

SET Class =2

WHERE Name = ‘Smith’

c. Insert a new course, <‘Knowledge Engineering’, ‘CS4390’, 3, ‘CS’>.

INSERT INTO COURSE

VALUES (‘Knowledge Engineering’, ‘CS4390’, 3, ‘CS’)

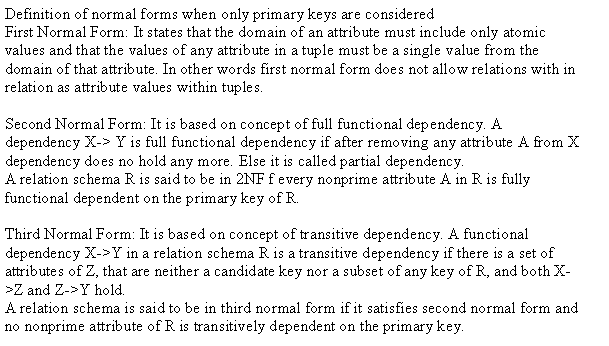
d. Delete the record for the student whose name is ‘Smith’ and whose student number is 17.

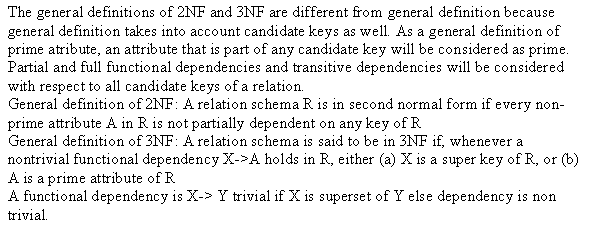
DELECT FROM STUDENT

WHERE Name =’ Smith’ AND Student\_number = 17

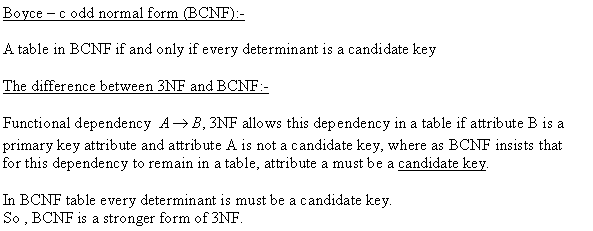
Chapter 15

15.8. Define first, second, and third normal forms when only primary keys are considered. How do the general definitions of 2NF and 3NF, which consider all keys of a relation, differ from those that consider only primary keys?





15.12. Define Boyce-Codd normal form. How does it differ from 3NF? Why is it considered a stronger form of 3NF?



15.18. Why do practical database designs typically aim for BCNF and not aim for higher normal forms?

Most practical design projects acquire existing designs of databases from previous

designs, designs in legacy models, or from existing files. Normalization is carried

out in practice so that the resulting designs are of high quality and meet the desirable properties stated previously. Although several higher normal forms have been

defined, such as the 4NF and 5NF that we discuss in Sections 15.6 and 15.7, the

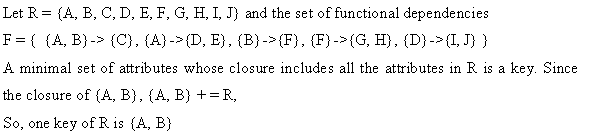
practical utility of these normal forms becomes questionable when the constraints

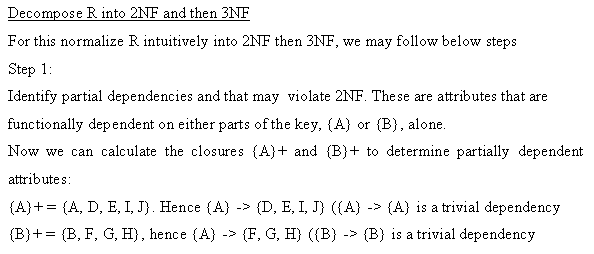
on which they are based are rare, and hard to understand or to detect by the database designers and users who must discover these constraints. Thus, database design

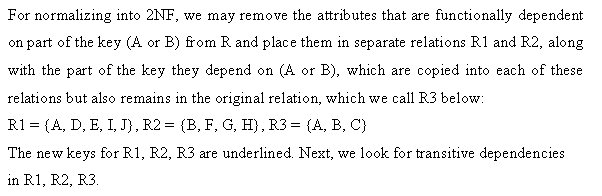
as practiced in industry today pays particular attention to normalization only up to

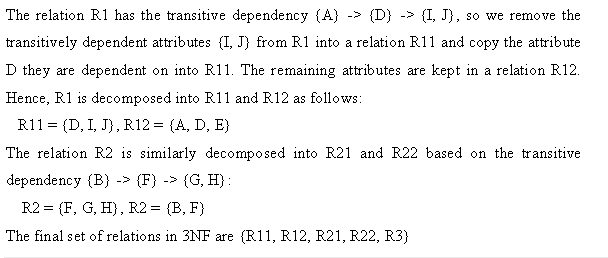
3NF, BCNF, or at most 4NF.

15.24. Consider the universal relation R= {A,B,C,D,E,F,G,H,I,J} and the set of functional dependencies F= { {A,B}→{C}, {A}→{D,E}, {B}→{F}, {F}→{G, H}, {D}→{I, J} }. What is the key for R? Decompose R into 2NF and then 3NF relations.









##########################################################

A minimal set of attributes whose closure includes all the attributes in R is a key. (One

can also apply algorithm 15.4a (see chapter 15 in the textbook)). Since the closure of

{A, B}, {A, B}+ = R, one key of R is {A, B} (in this case, it is the only key).

To normalize R intuitively into 2NF then 3NF, we take the following steps (alternatively,

we can apply the algorithms discussed in Chapter 15):

First, identify partial dependencies that violate 2NF. These are attributes that are

functionally dependent on either parts of the key, {A} or {B}, alone. We can calculate the

closures {A}+ and {B}+ to determine partially dependent attributes:

{A}+ = {A, D, E, I, J}. Hence {A} -> {D, E, I, J} ({A} -> {A} is a trivial dependency)

{B}+ = {B, F, G, H}, hence {A} -> {F, G, H} ({B} -> {B} is a trivial dependency)

To normalize into 2NF, we remove the attributes that are functionally dependent on part

of the key (A or B) from R and place them in separate relations R1 and R2, along with

the part of the key they depend on (A or B), which are copied into each of these relations

but also remains in the original relation, which we call R3 below:

R1 = {A, D, E, I, J}, R2 = {B, F, G, H}, R3 = {A, B, C}

The new keys for R1, R2, R3 are underlined. Next, we look for transitive dependencies

in R1, R2, R3. The relation R1 has the transitive dependency {A} -> {D} -> {I, J}, so we

remove the transitively dependent attributes {I, J} from R1 into a relation R11 and copy

the attribute D they are dependent on into R11. The remaining attributes are kept in a

relation R12. Hence, R1 is decomposed into R11 and R12 as follows:

R11 = {D, I, J}, R12 = {A, D, E}

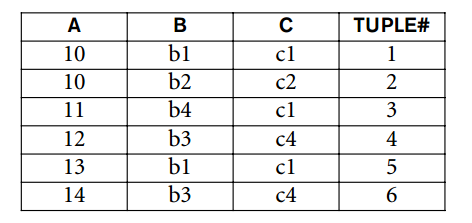
The relation R2 is similarly decomposed into R21 and R22 based on the transitive

dependency {B} -> {F} -> {G, H}:

R2 = {F, G, H}, R2 = {B, F}

The final set of relations in 3NF are {R11, R12, R21, R22, R3}

15.26. Consider the following relation:



a. Given the previous extension (state), which of the following dependencies

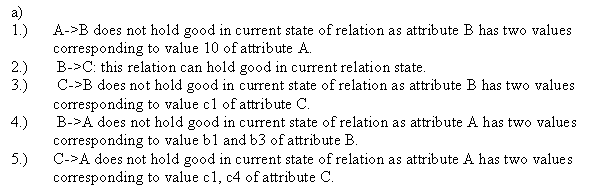
may holdin the above relation? If the dependency cannot hold, explain

why by specifying the tuples that cause the violation.

i.A→B, ii.B→C, iii.C→B,iv.B→A,v.C→A

b. Does the above relation have a potential candidate key? If it does, what is

it? If it does not, why not?





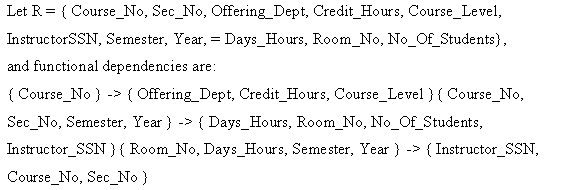


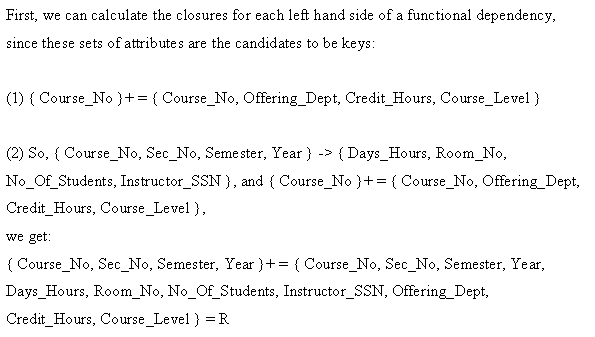
15.28. Consider the relation R, which has attributes that hold schedules of courses and sections at a university; R= {Course\_no, Sec\_no, Offering\_dept, Credit\_hours,Course\_level,Instructor\_ssn,Semester,Year,Days\_hours,Room\_no, No\_of\_students}. Suppose that the following functional dependencies hold on R: {Course\_no}→{Offering\_dept, Credit\_hours, Course\_level}

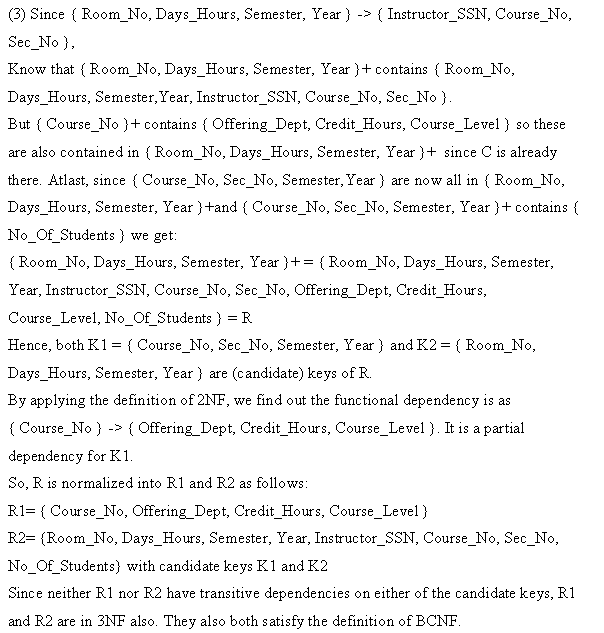
{Course\_no, Sec\_no, Semester, Year}→{Days\_hours, Room\_no, No\_of\_students, Instructor\_ssn}

{Room\_no, Days\_hours, Semester, Year}→{Instructor\_ssn, Course\_no, Sec\_no}

Try to determine which sets of attributes form keys of R. How would you normalize this relation?







##########################################################

Let us use the following shorthand notation:

C = CourseNo, SN = SecNo, OD = OfferingDept, CH = CreditHours, CL = CourseLevel,

I = InstructorSSN, S = Semester, Y = Year, D = Days\_Hours, RM = RoomNo,

NS = NoOfStudents

Hence, R = {C, SN, OD, CH, CL, I, S, Y, D, RM, NS}, and the following functional

dependencies hold:

{C} -> {OD, CH, CL}

{C, SN, S, Y} -> {D, RM, NS, I}

{RM, D, S, Y} -> {I, C, SN}

First, we can calculate the closures for each left hand side of a functional dependency,

since these sets of attributes are the candidates to be keys:

(1) {C}+ = {C, OD, CH, CL}

(2) Since {C, SN, S, Y} -> {D, RM, NS, I}, and {C}+ = {C, OD, CH, CL}, we get:

{C, SN, S, Y}+ = {C, SN, S, Y, D, RM, NS, I, OD, CH, CL} = R

(3) Since {RM, D, S, Y} -> {I, C, SN}, we know that {RM, D, S, Y}+ contains {RM, D, S,

Y, I, C, SN}. But {C}+ contains {OD, CH, CL} so these are also contained in {RM, D, S,

Y}+ since C is already there. Finally, since {C, SN, S, Y} are now all in {RM, D, S, Y}+

and {C, SN, S, Y}+ contains {NS} (from (2) above), we get:

{RM, D, S, Y}+ = {RM, D, S, Y, I, C, SN, OD, CH, CL, NS} = R

Hence, both K1 = {C, SN, S, Y} and K2 = {RM, D, S, Y} are (candidate) keys of R. By

applying the general definition of 2NF, we find that the functional dependency {C} ->

{OD, CH, CL} is a partial dependency for K1 (since C is included in K1). Hence, R is

normalized into R1 and R2 as follows:

R1 = {C, OD, CH, CL}

R2 = {RM, D, S, Y, I, C, SN, NS} with candidate keys K1 and K2

Since neither R1 nor R2 have transitive dependencies on either of the candidate keys, R1

and R2 are in 3NF also. They also both satisfy the definition of BCNF.

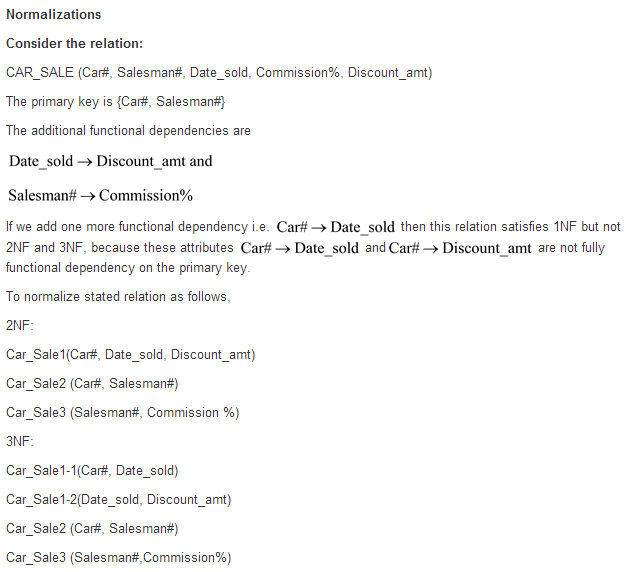
15.30. Consider the following relation:

CAR\_SALE(Car#, Date\_sold, Salesperson#, Commission%, Discount\_amt)

Assume that a car may be sold by multiple salespeople, and hence {Car#,Salesperson#} is the primary key. Additional dependencies are

Date\_sold→Discount\_amt and Salesperson# →Commission%

Based on the given primary key, is this relation in 1NF, 2NF, or 3NF? Why or why not? How would you successively normalize it completely?



###################################################

Given the relation schema

Car\_Sale(Car#, Salesman#, Date\_sold, Commission%, Discount\_amt)

with the functional dependencies

Date\_sold Discount\_amt

Salesman# Commission%

Car# Date\_sold

This relation satisfies 1NF but not 2NF (Car# Date\_sold and Car# 

Discount\_amt

so these two attributes are not FFD on the primary key) and not 3NF.

To normalize,

2NF:

Car\_Sale1(Car#, Date\_sold, Discount\_amt)

Car\_Sale2(Car#, Salesman#)

Car\_Sale3(Salesman#,Commission%)

3NF:

Car\_Sale1-1(Car#, Date\_sold)

Car\_Sale1-2(Date\_sold, Discount\_amt)

Car\_Sale2(Car#, Salesman#)

Car\_Sale3(Salesman#,Commission%)

15.31. Consider the following relation for published books:

BOOK (Book\_title, Author\_name, Book\_type, List\_price, Author\_affil, Publisher)

Author\_affil refers to the affiliation of author. Suppose the following dependencies exist:

Book\_title →Publisher, Book\_type

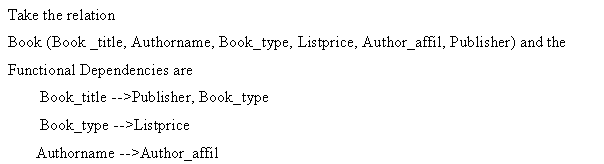
Book\_type →List\_price

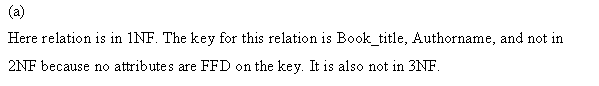
Author\_name→Author\_affil

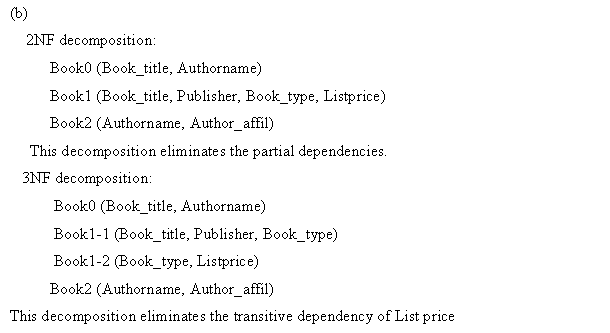
a. What normal form is the relation in? Explain your answer.

b. Apply normalization until you cannot decompose the relations further.

State the reasons behind each decomposition.







###############################################################

Given the relation

Book(Book\_title, Authorname, Book\_type, Listprice, Author\_affil, Publisher)

and the FDs

Book\_title Publisher, Book\_type

Book\_type Listprice

Authorname Author\_affil

(a)The key for this relation is Book\_title,Authorname. This relation is in 1NF and not in

2NF as no attributes are FFD on the key. It is also not in 3NF.

(b) 2NF decomposition:

Book0(Book\_title, Authorname)

Book1(Book\_title, Publisher, Book\_type, Listprice)

Book2(Authorname, Author\_affil)

This decomposition eliminates the partial dependencies.

3NF decomposition:

Book0(Book\_title, Authorname)

Book1-1(Book\_title, Publisher, Book\_type)

Book1-2(Book\_type, Listprice)

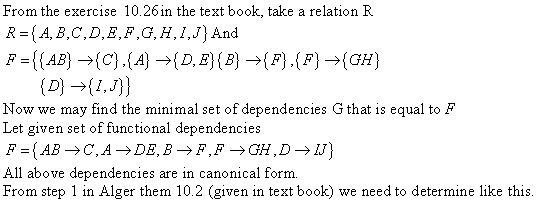
Book2(Authorname, Author\_affil)

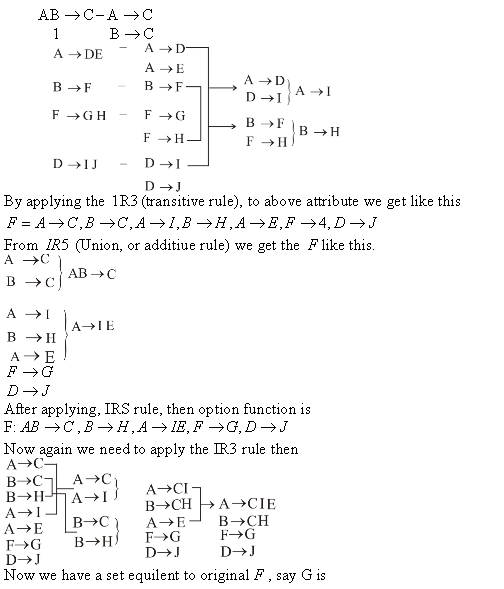
This decomposition eliminates the transitive dependency of Listprice

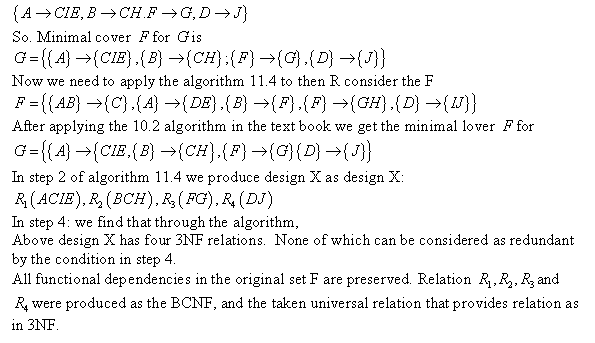
Chapter 16

16.26. Apply Algorithm 16.2(a) to the relation in Exercise 15.24 to determine a key for R. Create a minimal set of dependencies G that is equivalent to F, and apply the synthesis algorithm (Algorithm 16.6) to decompose R into 3NF relations.

(15.24. Consider the universal relation R= {A,B,C,D,E,F,G,H,I,J} and the set of functional dependencies F= { {A,B}→{C}, {A}→{D,E}, {B}→{F}, {F}→{G, H}, {D}→{I, J} }. What is the key for R? Decompose R into 2NF and then 3NF relations.)







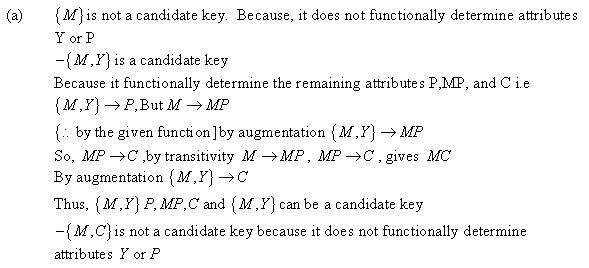
16.32. Consider the relation REFRIG(Model#,Year,Price,Manuf\_plant,Color), whichis abbreviated as REFRIG(M, Y, P, MP, C), and the following set Fof functional dependencies:F= {M→MP,{M,Y}→P,MP→C}

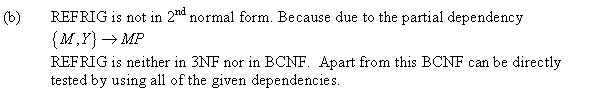
a.Evaluate each of the following as a candidate key for REFRIG, giving reasons why it can or cannot be a key: {M}, {M,Y}, {M,C}.

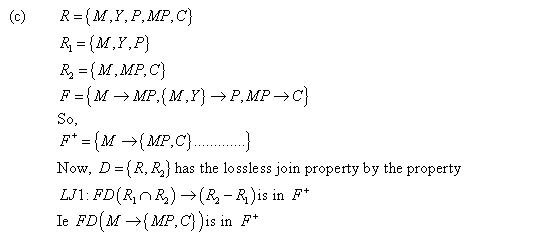
b.Based on the above key determination, state whether the relation REFRIGis in 3NF and in BCNF, giving proper reasons.

c.Consider the decomposition ofREFRIGinto D= {R1(M,Y,P),R2(M,MP,C)}. Is this decomposition lossless? Show why. (You may consult the testunder Property NJB in Section 16.2.4.)









####################################################################

(a)

- {M} IS NOT a candidate key since it does not functionally determine attributes Y or P.

- {M, Y} IS a candidate key since it functionally determines the remaining attributes P, MP, and C.

i.e.{M, Y} P, But M MP

By augmentation {M, Y} MP

Since MP C, by transitivity M MP, MP C, gives M C

By augmentation {M, Y} C

Thus {M, Y} P, MP, C and {M, Y} can be a candidiate key

- {M, C} IS NOT a candidate key since it does not functionally determine attributes Y or P.

(b)

REFRIG is not in 2NF, due to the partial dependency {M, Y} MP (since {M} MP

holds). Therefore REFRIG is neither in 3NF nor in BCNF.

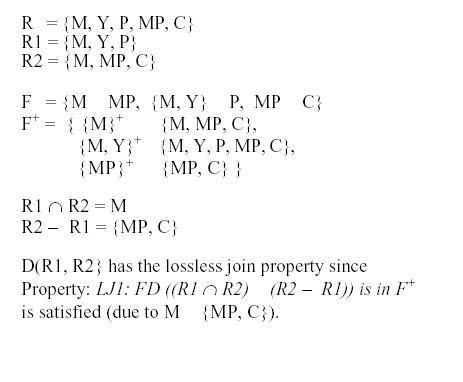
Alternatively: BCNF can be directly tested by using all of the given dependencies and

finding out if the left hand side of each is a superkey (or if the right hand side is a prime

attribute). In the two fields in REFRIG: M MP and MP C. Since neither M nor MP

is a superkey, we can conclude that REFRIG is is neither in 3NF nor in BCNF.

(c)



Chapter 21

21.9. What is a serial schedule? What is a serializable schedule? Why is a serial

schedule considered correct? Why is a serializable schedule considered correct?

Formally, a schedule S is serial if, for every transaction T participating in the schedule, all the operations of T are executed consecutively in the schedule; otherwise, the schedule is called nonserial. Therefore, in a serial schedule, only one transaction at a time is active—the commit (or abort) of the active transaction initiates execution of the next transaction. No interleaving occurs in a serial schedule. One reasonable assumption we can make, if we consider the transactions to be independent, is that every serial schedule is considered correct.

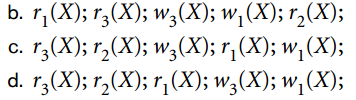
The definition of serializable schedule is as follows: A schedule S of n transactions is

Serializable if it is equivalent to some serial schedule of the same n transactions. We will define the concept of equivalence of schedules shortly. Notice that there are n! possible serial schedules of n transactions and many more possible nonserial schedules. We can form two disjoint groups of the nonserial schedules—those that are equivalent to one (or more) of the serial schedules and hence are serializable, and those that are not equivalent to any serial schedule and hence are not serializable.

We can assume this because every transaction is assumed to be correct if executed on its own (according to the consistency preservation property of Section 21.3). Hence, it does not matter which transaction is executed first. As long as every transaction is executed from beginning to end in isolation from the operations of other transactions, we get a correct end result on the database.

Saying that a nonserial schedule S is serializable is equivalent to saying that it is correct, because it is equivalent to a serial schedule, which is considered correct.

21.22. Which of the following schedules is (conflict) serializable? For each serializable schedule, determine the equivalent serial schedules.



Let there be three transactions T1, T2, and T3. They are executed concurrently and produce a schedule S. S is serializable if it can be reproduced as at least one serial schedule (T1 T2

T3 or T1 T3 T2 or T2 T1 T3 or T2 T3 T1 or T3 T1 T2 or T3 T2

T1).

(b) This schedule is not serializable because T1 reads X ( r1(X)) before T3 but T3 writes X

(w3(X)) before T1 writes X (w1(X)). The operation r2(X) of T2 does not affect the

schedule at all so its position in the schedule is irrelevant. In a serial schedule T1, T3,

and T2, r3(X) and w3(X) must come after w1(X), which does not happen in the question.

(c) This schedule is **serializable** because all conflicting operations of T3 happens before

all conflicting operation of T1. T2 has only one operation, which is a read on X (r2(X)),

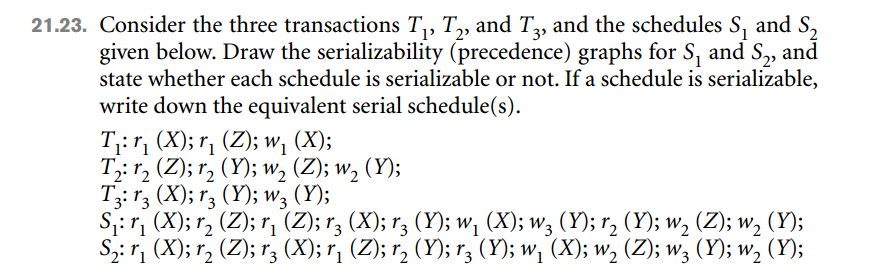
which does not conflict with any other operation. Thus this serializable schedule is

equivalent to r2(X); r3(X); w3(X); r1(X); w1(X) (e.g., T2 T3 T1) serial schedule.

(d) This is not a serializable schedule because T3 reads X (r3(X)) before T1 reads X (r1(X))

but r1(X) happens before T3 writes X (w3(X)). In a serial schedule T3, T2, and T1, r1(X)

will happen after w3(X), which does not happen in the question.



Schedule S1: It is a serializable schedule because

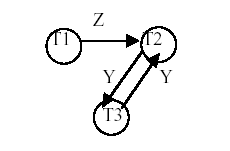
* T1 only reads X (r1(X)), which is not modified either by T2 or T3,
* T3 reads X (r3(X)) before T1 modifies it (w1(X)),
* T2 reads Y (r2(Y)) and writes it (w2(Y)) only after T3 has written to it (w3(Y))
* Thus, the serializability graph is

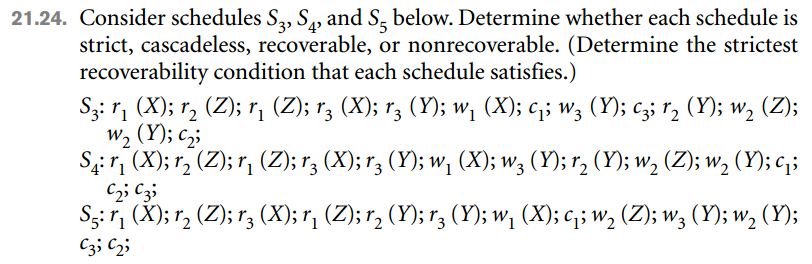


Schedule is not a serializable schedule because

* T2 reads Y (r2(Y)), which is then read and modified by T3 (w3(Y))
* T3 reads Y (r3(Y)), which then modified before T2 modifies Y (w2(Y))

In the above order T3 interferes in the execution of T2, which makes the schedule nonserializable.





**Strict schedule**: A schedule is strict if it satisfies the following conditions:

1. Tj reads a data item X ***after*** Ti has written to X and Ti is terminated (aborted or

committed)

2. Tj writes a data item X ***after*** Ti has written to X and Ti is terminated (aborted or

committed)

**Schedule S3 is not strict** because T3 reads X (r3(X)) ***before*** T1 has written to X (w1(X))

but T3 commits ***after*** T1. In a strict schedule T3 must read X ***after*** C1.

**Schedule S4 is not strict** because T3 reads X (r3(X)) ***before*** T1 has written to X (w1(X))

but T3 commits ***after*** T1. In a strict schedule T3 must read X ***after*** C1.

**Schedule S5 is not strict** because T3 reads X (r3(X)) ***before*** T1 has written to X (w1(X))

but T3 commits ***after*** T1. In a strict schedule T3 must read X ***after*** C1.

**Cascadeless schedule**: A schedule is cascadeless if the following condition is satisfied:

Tj reads X only ***after*** Ti has written to X and terminated (aborted or committed).

Schedule S3 is ***not cascadeless*** because T3 reads X (r3(X)) before T1 commits.

Schedule S4 is ***not cascadeless*** because T3 reads X (r3(X)) before T1 commits.

Schedule S5 is ***not cascadeless*** because T3 reads X (r3(X)) ***before*** T1 commits or T2 reads

Y (r2(Y)) ***before*** T3 commits.

**NOTE**: According to the definition of cascadeless schedules S3, S4, and S4 are not

cascadeless. However, T3 is not affected if T1 is rolled back in any of the schedules, that is,

T3 does not have to roll back if T1 is rolled back. The problem occurs because these

schedules are not serializable.

**Recoverable schedule**: A schedule is recoverable if the following condition is satisfied:

Tj commits after Ti if Tj has read any data item written by Ti.

NOTE: Ci > Cj means Ci happens ***before*** Cj. Ai denotes abort Ti. To test if a schedule is

recoverable one has to include abort operations. Thus in testing the recoverability abort

operations will have to used in place of commit one at a time. Also the strictest condition is

where a transaction neither reads nor writes to a data item, which was written to by a

transaction that has not committed yet.

If A1>C3>C2, then S3 is ***recoverable*** because rolling back of T1 does not affect T2 and

T3. If C1>A3>C2. S3 is ***not recoverable*** because T2 read the value of Y (r2(Y)) ***after***

T3 wrote X (w3(Y)) and T2 committed but T3 rolled back. Thus, T2 used non- existent

value of Y. If C1>C3>A3, then S3 is ***recoverable*** because roll back of T2 does not

affect T1 and T3. Strictest condition of S3 is C3>C2.

If A1>C2>C3, then S4 is ***recoverable*** because roll back of T1 does not affect T2 and

T3. If C1>A2>C3, then S4 is ***recoverable*** because the roll back of T2 will restore the

value of Y that was read and written to by T3 (w3(Y)). It will not affect T1. If

C1>C2>A3, then S4 is ***not recoverable*** because T3 will restore the value of Y which was

not read by T2. Strictest condition of S4 is C3>C2, but it is not satisfied by S4.

If A1>C3>C2, then S5 is ***recoverable*** because neither T2 nor T3 writes to X, which is

written by T1. If C1>A3>C2, then S5 is ***not recoverable*** because T3 will restore the

value of Y, which was not read by T2. Thus, T2 committed with a non-existent value of

Y. If C1>C3>A2, then S5 is ***recoverable*** because it will restore the value of Y to the

value, which was read by T3. Thus, T3 committed with the right value of Y. Strictest

condition of S3 is C3>C2, but it is not satisfied by S5.