Selected Exercises from Fundamentals of Database Systems

Andrew Kowalczyk

I selected around 40 questions from the 6^{th} edition of *Fundamentals of Database Systems* by Ramez Elmasri and Shamkant Navathe. This was part of my requirement for my Introduction to Database Systems course taken in the Fall of 2013.

Contents

Chapter 1

1.8 Identify some informal queries and update operations that you would expect to apply to the database shown in Figure 1.2.

Queries

1. What are the prerequisites of the Database course?
2. Find the names of all students majoring in Mathematics.
3. Find the the transcript of the student named Brown.

Updates

1. Insert a new student in the database whose Name=Kowalczyk, Student_number=25, Class=4, and Major=CS.
2. Change the grade that Brown received in Discrete Mathematics to a D.

1.9 What is the difference between controlled and uncontrolled redundancy? Illustrate with examples.

Redundancy is the term given when the same data is stored multiple times in several places in a database. If you look at Figure 1.5(a) in the text, you can see that the name of the student with Student_number=8 is Brown is stored multiple times. Redundancy is controlled when the database management system (DBMS) ensures that multiple copies of the same data are consistent. To illustrate this, let's say we are adding a new record with Student_number=8 to be stored in the database of Figure 1.5(a). If we were to have uncontrolled redundancy, the DBMS would have no control over this. If we were to have controlled redundancy, the DBMS would ensure that Student_name=Brown in that record.

1.10 Specify all the relationships among the records of the database shown in Figure 1.2.

1. Every GRADE_REPORT record is related to one STUDENT record and one SECTION record.

2. Every SECTION record is related to a COURSE record.

3. Every PREREQUISITE record relates two COURSE records. One being a course and the other being a prerequisite to that course.

1.11 Give some additional views that may be needed by other user groups for the database shown in Figure 1.2.

1. A view of each class section that groups all the students who took that section and their respective grade.

2. A view that gives the number of courses taken and the grade point average for each student.

1.12 Cite some examples of integrity constraints that you think can apply to the database shown in Figure 1.2.

Key constraints

1. Student_number must be unique for each STUDENT record.

2. Course_number must be unique for each COURSE record.

Referential integrity constraints

1. The Course_number in a SECTION record must also exist in some COURSE record.

2. The Student_number in a GRADE_REPORT record must also exist in some STUDENT record.

Domain constraints

1. Grades in a a given GRADE_REPORT record must be one of these values: A, B, C, D, F, I, W.

Chapter 2

2.12 Think of different users for the database shown in Figure 1.2. What types of applications would each user need? To which user category would each belong, and what type of interface would each need?

1. Students add and drop classes. Actions that they can do are as listed:
   (a) Register themselves in a section of a course
   (b) Drop themselves from a section of a course
2. Registrar. They enter data of registration of students in sections of courses, and later enter the grades of the students. Actions that they can do are as listed:
   (a) Check whether a student who is registered in a course has the appropriate prerequisite courses
   (b) Add a student to a section of a course
   (c) Enter the student grades for a section
3. Admissions. Their main application would be to enter newly accepted students into the database. Actions that they can do are as listed:
   (a) Add students to the school's records

Chapter 3

3.11 Suppose that each of the following Update operations is applied directly to the database state shown in Figure 3.6. Discuss all integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints.

In each of examples, the different ways of enforcing the constraints is listed in preferential order (*i.e.* 1 is most preferred, 2 is less preferred, etc.).

(a) Insert <"Robert", "F", "Scott", "943775543", "1972-06-21", "2365 Newcastle Rd, Bellaire, TX", M, 58000, "888665555", 1> into EMPLOYEE.

No constraints violated.

(b) Insert <"ProductA", 4, "Bellaire", 2> into PROJECT.

Since there is no tuple in the DEPARTMENT relation with DNUM=2, this insertion would violate referential integrity for that very reason. We can enforce the constraint with these options:

1. Rejecting the insertion
2. Changing the value of DNUM in the new PROJECT tuple to an existing DNUMBER value in the DEPARTMENT relation
3. Inserting a new DEPARTMENT tuple with DNUMBER=2

(c) Insert <"Production", 4, "943775543", "2007-10-01"> into DEPARTMENT.

Oh no! Since there already exists a DEPARTMENT tuple with DNUMBER=4, this would violate the key constraint. Enforcement of this constraint would happen by:

1. Rejecting the insertion
2. Changing the value of DNUMBER in the new DEPARTMENT tuple to a value that does not violate the key constraint

Also, since there is no tuple in the EMPLOYEE relation with SSN="943775543", this insertion also happens to violate referential integrity. Let us enforce the constraint by either:

1. Rejecting the insertion
2. Changing the value of MGRSSN to an existing SSN value in EMPLOYEE
3. Inserting a new EMPLOYEE tuple with SSN="943775543"

(d) Insert <"677678989", NULL, "40.0"> into WORKS_ON.

Since PNO (part of the primary key of WORKS_ON) is null, this violates entity integrity. We have these options to enforce this constraint:

1. Rejecting the insertion
2. Changing the value of PNO in the new WORKS_ON tuple to a value of PNUMBER that exists in the PROJECT relation

Since there is no tuple in the EMPLOYEE relation with SSN="677678989", this insertion would violate referential integrity. We may enforce the constraint by:

1. Rejecting the insertion
2. Changing the value of ESSN to an existing SSN value in EMPLOYEE
3. Inserting a new EMPLOYEE tuple with SSN="677678989"

(e) Insert <'453453453','John','M','1990-12-12','spouse'> into DEPENDENT.

Nope! Not a single constraint was violated.

(f) Delete the WORKS_ON tuples with Essn = '333445555'.

No constraints violated in the making of this delete.

(g) Delete the EMPLOYEE tuple with Ssn = '987654321'.

Unfortunately, the employee trying to be deleted is referenced in the WORKS_ON, DEPENDENT, DEPARTMENT, and EMPLOYEE relations. We can enforce such an abolishment by either:

1. Rejecting the deletion
2. Deleting all tuples in the WORKS_ON, DEPENDENT, DEPARTMENT, and EMPLOYEE relations whose values for ESSN, ESSN, MGRSSN, and SUPERSSN, respectively, is equal to '987654321'

(h) Delete the PROJECT tuple with Pname='ProductX'.

This deletion would completely violate referential integrity because two tuples exist in the WORKS_ON relation that reference the tuple being deleted from PROJECT. Silly us, let's enforce the constraint by:

1. Rejecting the deletion
2. Deleting the tuples in the WORKS_ON relation whose value for the primary key PNUMBER for the tuple being deleted from PROJECT with PNO=1

(i) Modify the Mgr_ssn and Mgr_start_date of the DEPARTMENT tuple with Dnumber = 5 to '123456789' and '2007-10-01', respectively.

No violation of constraints.

(j) Modify the Super_ssn attribute of the EMPLOYEE tuple with Ssn = '999887777' to '943775543'.

Goodness gracious, great balls of fire! Since there is no tuple in the EMPLOYEE relation with SSN='943775543', this update would violate referential integrity. In order to enforce this constraint, we can choose from:

1. Rejecting the deletion
2. Inserting a new EMPLOYEE tuple with SSN='943775543'

(k) Modify the Hours attribute of the WORKS_ON tuple with Essn = '999887777' and Pno = 10 to '5.0'.

No constraints violated.

Chapter 4

4.9 How can the key and foreign key constraints be enforced by the DBMS? Is the enforcement technique you suggest difficult to implement? Can the constraint checks be executed efficiently when updates are applied to the database?

In order to check efficiently for the key constraint, one can create an index on all of the attributes that form each primary or secondary key. Even before the new record is inserted, each index is searched to insure that no value currently exists in the index that matches the key value in the new record. If this is the case, the record is inserted successfully and we are happy.

In order to check efficiently for the foreign key constraint, the primary key is given an index for each referenced relation. Due to this, the check efficient enough for our purposes. Each time a new record is inserted in a referencing relation, we search the index for the primary key of the referenced relation. The value of its foreign key is used to accomplish this. The new record can be successfully inserted in the referencing relation if the referenced record exists. For the deletion of any given referenced record, an index on the foreign key of each of the referencing relations is very useful. We want to efficiently determine whether any records reference that given record. If the techniques described above do not exist, then, unfortunately, we must do linear searches to check for any of the above constraints. This would making the checks quite inefficient, and it would make us unhappy.

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 sci- ence).

```sql
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.

```sql
SELECT Course_name
FROM   COURSE,
       SECTION
WHERE COURSE.Course_number=SECTION.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.

```sql
SELECT Course_number,
       Semester,
       Year,
```

```
        COUNT(*)
FROM    SECTION,
        GRADE_REPORT
WHERE Instructor="King"
    AND SECTION.Section_identifier=GRADE_REPORT.Section_identifier
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,
        C.Course_name,
        C.Course_number,
        Credit_hours,
        Semester,
        Year,
        Grade
FROM    STUDENT ST,
        COURSE C,
        SECTION S,
        GRADE_REPORT G
WHERE Class=4
    AND Major="CS"
    AND ST.StudentNumber=G.StudentNumber
    AND G.Section_identifier=S.Section_identifier
    AND S.Course_number=C.Course_number
```

Chapter 6

6.15 Show the result of each of the sample queries in Section 6.5 as it would apply to the database state in Figure 3.6.

Query 1: Find the name and address of all employees who work for the 'Research' department.

FNAME	LNAME	ADDRESS
John	Smith	731 Fondren, Houston, TX
Franklin	Wong	638 Voss, Houston, TX
Ramesh	Narayan	975 Fire Oak, Humble, TX
Joyce	English	5631 Rice, Houston, TX

Query 2: For every project located in 'Stafford', list the project number, the controlling department number, and the department manager's last name, address, and birth date.

PNUMBER	DNUM	LNAME	ADDRESS	BDATE
10	4	Wallace	291 Berry, Bellaire, TX	20-JUN-31
30	4	Wallace	291 Berry, Bellaire, TX	20-JUN-31

Query 3: Find the names of all employees who work on all the projects controlled by department number 5.

LNAME	FNAME

Query 4: Make a list of project numbers for projects that involve an employee whose last name is 'Smith' as a worker or as a manager of the department that controls the project.

PNO
1
2

Query 5: List the names of all employees with two or more dependents.

LNAME	FNAME
Smith	John
Wong	Franklin

Query 6: List the names of employees who have no dependents.

LNAME	FNAME
Zelaya	Alicia
Narayan	Ramesh
English	Joyce
Jabbar	Ahmad
Borg	James

Query 7: List the names of managers who have at least one dependent.

LNAME	FNAME
Wallace	Jennifer
Wong	Franklin

6.16 Specify the following queries on the COMPANY relational database schema shown in Figure 5.5, 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.

I use the symbol σ for SELECT, π for PROJECT, $\triangleright\triangleleft$ for EQUIJOIN, * for NATURAL JOIN, and $\mathfrak{F}$ for FUNCTION.

(a) Retrieve the names of employees in department 5 who work more than 10 hours per week on the "ProductX" project.

Relational Operators

EMP_W_X $\Leftarrow$ (σ $_{\text{PNAME="ProductX"}}$ (PROJECT)) $\triangleright\triangleleft$ $_{\text{(PNUMBER),(PNO)}}$ (WORKS_ON)
EMP_WORK_10 $\Leftarrow$ (EMPLOYEE) $\triangleright\triangleleft$ $_{\text{(SSN),(ESSN)}}$ (σ $_{\text{HOURS>10}}$ (EMP_W_X))
RESULT $\Leftarrow$ π $_{\text{LNAME,FNAME}}$ (σ $_{\text{DNO=5}}$ (EMP_WORK_10))

Result

LNAME	FNAME
Smith	John
English	Joyce

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

Relational Operators

E $\Leftarrow$ (EMPLOYEE) $\triangleright\triangleleft$ $_{\text{(SSN,FNAME),(ESSN,DEPENDENT\_NAME)}}$ (DEPENDENT)
R $\Leftarrow$ π $_{\text{LNAME,FNAME}}$ (E)

Result

LNAME	FNAME

(c) Find the names of employees that are directly supervised by "Franklin Wong".

Relational Operators

WONG_SSN $\Leftarrow \pi_{\text{SSN}} (\sigma_{\text{FNAME="Franklin" AND LNAME="Wong"}}$ (EMPLOYEE))
WONG_EMPS $\Leftarrow$ (EMPLOYEE) $\rhd\lhd_{\text{(SUPERSSN),(SSN)}}$ (WONG_SSN)
RESULT $\Leftarrow \pi_{\text{LNAME,FNAME}}$ (WONG_EMPS)

Result

LNAME	FNAME
Smith	John
Narayan	Ramesh
English	Joyce

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

Relational Operators

PROJ_HOURS(PNO,TOT_HRS) $\Leftarrow {}_{\text{PNO}} \mathfrak{F}_{\text{SUM HOURS}}$ (WORKS_ON)
RESULT $\Leftarrow \pi_{\text{PNAME,TOT\_HRS}}$ ((PROJ_HOURS) $\rhd\lhd_{\text{(PNO),(PNUMBER)}}$ (PROJECT))

Result

PNAME	TOT_HRS
ProductX	52.5
ProductY	37.5
ProductZ	50.0
Computerization	55.0
Reorganization	25.0
Newbenefits	55.0

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

Relational Operators

PROJ_EMPS(PNO,SSN) $\Leftarrow \pi_{\text{PNO,ESSN}}$ (WORKS_ON)
ALL_PROJS(PNO) $\Leftarrow \pi_{\text{PNUMBER}}$ (PROJECT)

$\text{EMPS\_ALL\_PROJS} \Leftarrow \text{PROJ\_EMPS} \div \text{ALLPROJS}$
$\text{RESULT} \Leftarrow \pi_{\text{LNAME,FNAME}}\ (\text{EMPLOYEE} * \text{EMP\_ALL\_PROJS})$

Result

LNAME	FNAME

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

Relational Operators

$\text{ALL\_EMPS} \Leftarrow \pi_{\text{SSN}}\ (\text{EMPLOYEE})$
$\text{WORKING\_EMPS(SSN)} \Leftarrow \pi_{\text{ESSN}}\ (\text{WORKS\_ON})$
$\text{NON\_WORKING\_EMPS} \Leftarrow \text{ALL\_EMPS - WORKING\_EMPS}$
$\text{RESULT} \Leftarrow \pi_{\text{LNAME,FNAME}}\ (\text{EMPLOYEE} * \text{NON\_WORKING\_EMPS})$

Result

LNAME	FNAME

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

Relational Operators

$\text{DEPT\_AVG\_SALS(DNUMBER,AVG\_SAL)} \Leftarrow {}_{\text{DNO}}\,\mathfrak{F}_{\text{AVG SALARY}}\ (\text{EMPLOYEE})$
$\text{RESULT} \Leftarrow \pi_{\text{DNUMBER,AVG\_SAL}}\ (\ \text{DEPT\_AVG\_SALS} * \text{DEPARTMENT}\ )$

Result

DNUMBER	AVG_SAL
Research	33250
Administration	31000
Headquarters	55000

(h) Retrieve the average salary of all female employees.

Relational Operators

$\text{RESULT(AVG\_F\_SAL)} \Leftarrow \mathfrak{F}_{\text{AVG SALARY}}\ (\ \sigma_{\text{SEX="F"}}\ (\text{EMPLOYEE})\ )$

Result

AVG_F_SAL
31,000

(i) 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.

Relational Operators

E_P_HOU(SSN) $\Leftarrow \pi_{\text{ESSN}}$ (WORKS_ON $\rhd\lhd_{\text{(PNO),(PNUMBER)}}$ ($\sigma_{\text{PLOCATION="Houston"}}$ (PROJECT)))
D_NO_HOU $\Leftarrow \pi_{\text{DNUMBER}}$ (DEPARTMENT) - π_{DNUMBER} ($\sigma_{\text{DLOCATION="Houston"}}$ (DEPARTMENT))
E_D_NO_HOU $\Leftarrow \pi_{\text{SSN}}$ (EMPLOYEE $\rhd\lhd_{\text{(PNO),(DNUMBER)}}$ (D_NO_HOU))
RESULT_EMPS $\Leftarrow$ E_P_HOU - E_D_NO_HOU
RESULT $\Leftarrow \pi_{\text{LNAME,FNAME,ADDRESS}}$ (EMPLOYEE * RESULT_EMPS)

Result

LNAME	FNAME	ADDRESS
Wallace	Jennifer	291 Berry, Bellaire, TX

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

Relational Operators

DEPT_MANAGERS(SSN)<– π_{MGRSSN} (DEPARTMENT)
EMPS_WITH_DEPENDENTS(SSN) $\Leftarrow \pi_{\text{ESSN}}$ (DEPENDENT)
RESULT_EMPS $\Leftarrow$ DEPT_MANAGERS - EMPS_WITH_DEPENDENTS
RESULT $\Leftarrow \pi_{\text{LNAME,FNAME}}$ (EMPLOYEE * RESULT_EMPS)

Relational Operators

LNAME	FNAME
Borg	James

Chapter 7

7.16 Consider the following set of requirements for a UNIVERSITY database that is used to keep track of students? transcripts. This is similar but not identical to the database shown in Figure 1.2.

(a) The university keeps track of each student's name, student number, Social Security number, current address and phone number, permanent address and phone number, birth date, sex, class (freshman, sophomore, ..., graduate), major department, minor department (if any), and degree program (B.A., B.S., ..., Ph.D.). Some user applications need to refer to the city, state, and ZIP Code of the student's permanent address and to the student's last name. Both Social Security number and student number have unique values for each student.

(b) Each department is described by a name, department code, office number, office phone number, and college. Both name and code have unique values for each department.

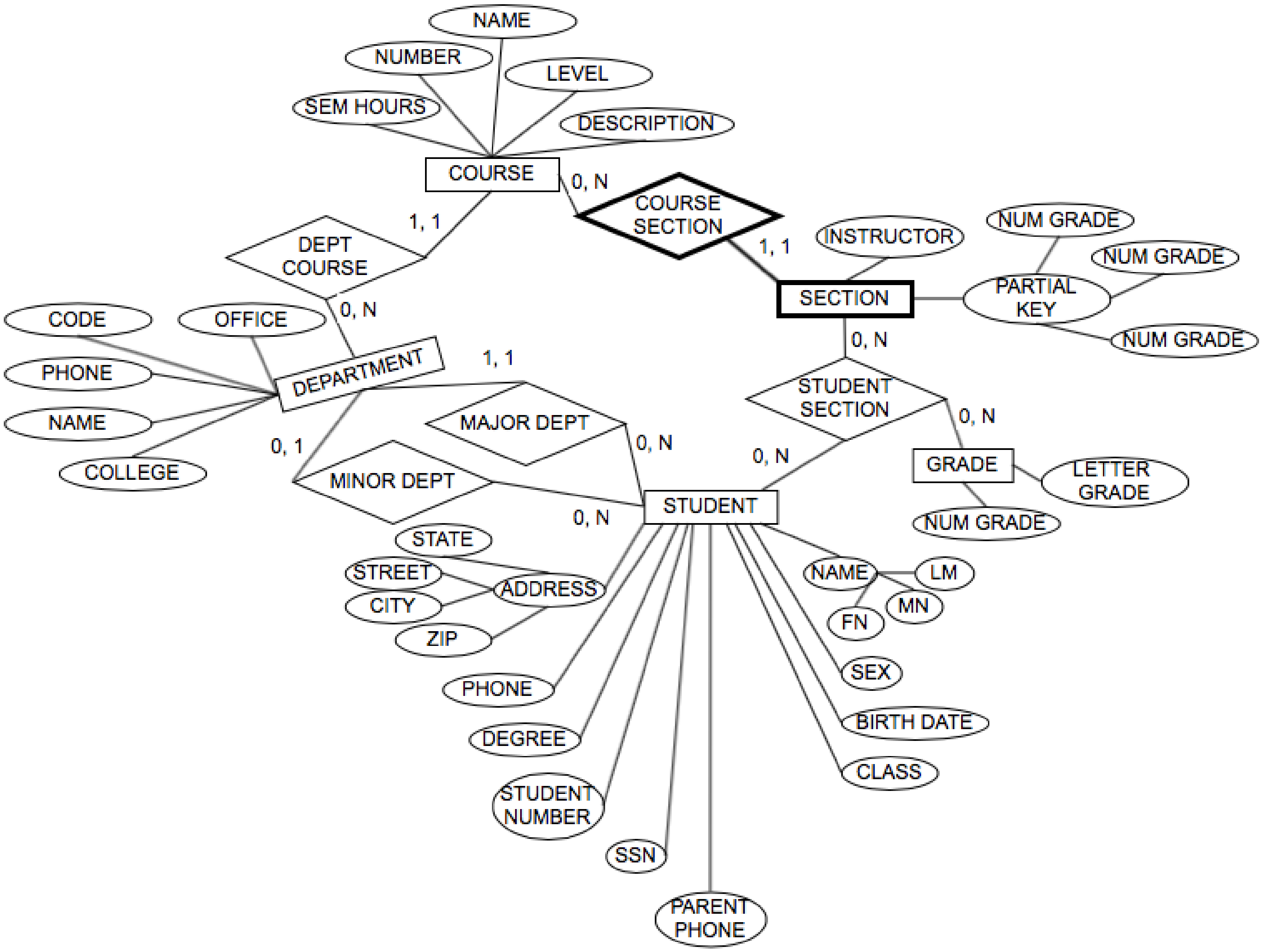
(c) Each course has a course name, description, course number, number of semester hours, level, and offering department. The value of the course number is unique for each course.

(d) Each section has an instructor, semester, year, course, and section number. The section number distinguishes sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ..., up to the number of sections taught during each semester.

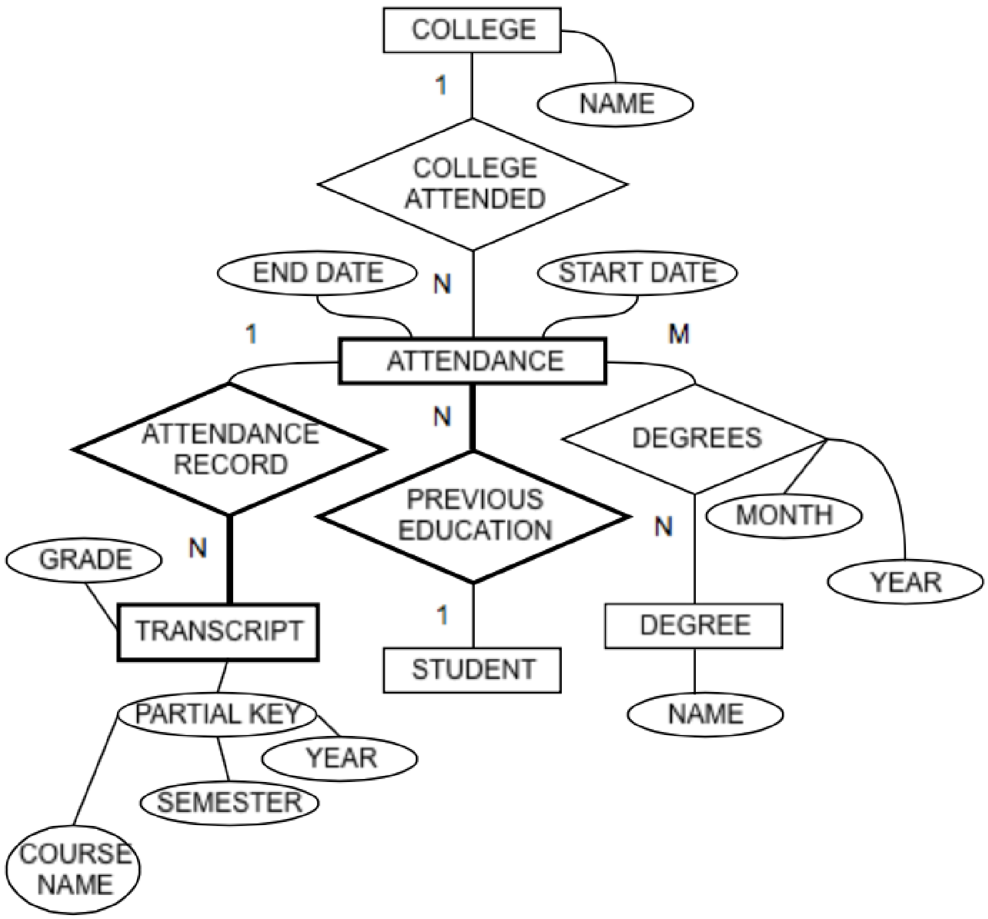
(e) A grade report has a student, section, letter grade, and numeric grade (0, 1, 2, 3, or 4).

Design an ER schema for this application, and draw an ER diagram for the schema. Specify key attributes of each entity type, and structural constraints on each relationship type. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.

7.16 (cont)


NAME
NUMBER
LEVEL
SEM HOURS
DESCRIPTION
COURSE
0, N
COURSE SECTION
1, 1
DEPT COURSE
1, 1
INSTRUCTOR
NUM GRADE
NUM GRADE
PARTIAL KEY
NUM GRADE
SECTION
0, N
CODE
OFFICE
PHONE
DEPARTMENT
1, 1
0, N
NAME
COLLEGE
STUDENT SECTION
0, N
MAJOR DEPT
0, N
0, 1
MINOR DEPT
0, N
GRADE
LETTER GRADE
NUM GRADE
STUDENT
0, N
STATE
STREET
ADDRESS
CITY
ZIP
NAME
LM
MN
FN
SEX
PHONE
BIRTH DATE
DEGREE
CLASS
STUDENT NUMBER
SSN
PARENT PHONE

7.18 Show an alternative design for the attribute described in Exercise 7.17 that uses only entity types (including weak entity types, if needed) and relationship types.



7.19 Consider the ER diagram in Figure 7.20, which shows a simplified schema for an airline reservations system. Extract from the ER diagram the requirements and constraints that produced this schema. Try to be as precise as possible in your requirements and constraints specification.

1. The given database represents any given AIRPORT. It keeps its unique Airport_code, AIRPORT name, city and state in which it is located.

2. For each LEG_INSTANCE, the customer RESERVATIONs includes a Customer_name, Cphone, and Seat_no(s).

3. One can see that each FLIGHT has its unique number, its Airline, and the Weekdays on which it is scheduled.

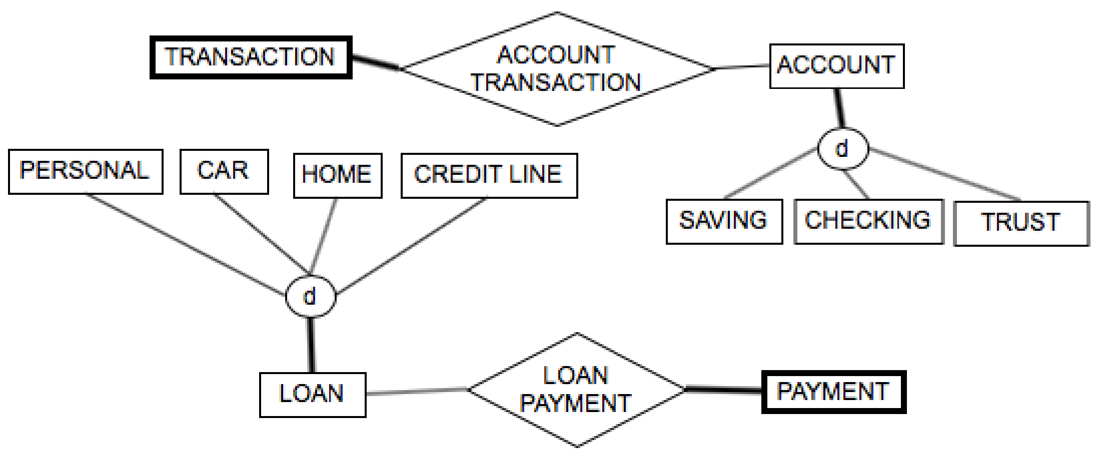
4. Information on AIRPLANEs and AIRPLANE TYPEs are also kept. Every AIRPLANE TYPE stores the Type_name, manufacturing Company, and Maximum Number of Seats, and AIRPORTs in which planes of that type CAN_LAND. For each AIRPLANE, the airplane ID, Total number of seats, and TYPE are kept.

5. On a specific Date, a LEG_INSTANCE is an instance of a FLIGHT_LEG. After any FLIGHT_LEG has been concluded, the DEPARTURE_AIRPORT, ARRIVAL_AIRPORT, the arrival times, number of available seats, and the AIRPLANE used are recorded for that FLIGHT_LEG.

6. Any given FLIGHT has one or more FLIGHT_LEGs. Each FLIGHT_LEG has a DEPARTURE_AIRPORT and ARRIVAL_AIRPORT, with their respective Scheduled_dep_time and Scheduled_arr_time.

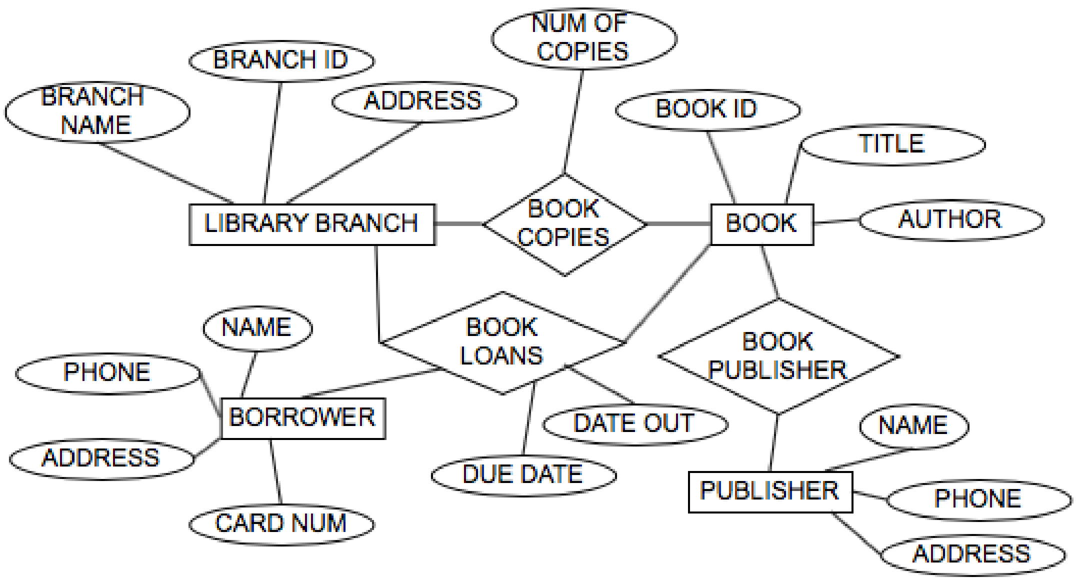
Chapter 8

8.17 Consider the BANK ER schema in Figure 7.21, and suppose that it is necessary to keep track of different types of ACCOUNTS (SAVINGS_ACCTS, CHECKING_ACCTS, ...) and LOANS (CAR_LOANS, HOME_LOANS, ...). Suppose that it is also desirable to keep track of each ACCOUNT's TRANSACTIONS (deposits, withdrawals, checks, ...) and each LOAN's PAYMENTS; both of these include the amount, date, and time. Modify the BANK schema, using ER and EER concepts of specialization and generalization. State any assumptions you make about the additional requirements.



Chapter 9

9.3 Try to map the relational schema in Figure 6.14 into an ER schema. This is part of a process known as reverse engineering, where a conceptual schema is created for an existing implemented database. State any assumptions you make.



Book authors, in this particular diagram, are represented as a multi-valued attribute of books.

Chapter 11

11.31 Map the COMPANY ER schema in Figure 7.2 into ODL classes. Include appropriate methods for each class.

class Employee

(**extent** employees

key ssn)

{ **attribute** **struct** name { **string** fname,

```
                                   string mname,
                                   string lname  } name;
  attribute     string           ssn;
  attribute     date             bdate;
  attribute     enum             GenderM, F sex;
  attribute     string           address;
  attribute     float            salary;
  attribute     Employee         supervisor;
  relationship  Department works_for inverse Department:: has_employees;
  relationship  set<Hours_Worked> work inverse Hours_Worked:: work_by;
  short         age();
  void          give_raise(in float raise);
  void          change_address(in string new_address);
  void          reassign_emp(in string new_dname) raises (dname_not_valid); };

class Department
( extent        departments
  key           dname, dnumber )
{ attribute     string           dname;
  attribute     short            dnumber;
  attribute     struct           Dept_Mgr Employee manager, date startdate mgr;
  attribute     set<string>      locations;
  relationship  set<Employee> has_employees inverse Employee:: works_for;
  relationship  set<Project> controls inverse Project:: controlled_by;
  short         no_of_employees();
  void          add_emp(in string new_essn) raises (essn_not_valid);
  void          add_proj(in string new_pname) raises (pname_not_valid);
  void          change_manger(in string new_mssn; in date startdate) raises (mssn_not_
```

```
class Project
( extent        projects
  key           pname, pnumber )
{ attribute     string        pname;
  attribute     short         pnumber;
  attribute     string        location;
  relationship  Department controlled_by inverse Department:: controls;
  relationship  set<Hours_Worked> work_at inverse Hours_Worked:: work_on;
  void          reassign_proj(in string new_dname) raises (dname_not_valid) };

class Hours_Worked
( extent        hours_worked )
{ attribute     float         hours;
  relationship  Employee work_by inverse Employee:: work;
  relationship  Project work_on inverse Project:: work_at;
  void          change_hours(in float new_hours); };

class Dependent
( extent        dependents )
{ attribute     string        name;
  attribute     date          bdate;
  attribute     enum          Gender{M, F} sex;
  attribute     string        relationship;
  attribute     Employee      suporter;
  short         age();   };
```

Chapter 15

15.21 In what normal form is the LOTS relation schema in Figure 15.12(a) with respect to the restrictive interpretations of normal form that take only the primary key into account? Would it be in the same normal form if the general definitions of normal form were used?

By only taking the primary key into account, we can see that the relation schema will be in 2NF since there are no partial dependencies on the primary key. However, our relation schema is not in 3NF due to the fact that the primary key has two transitive dependencies:

PROPERTY_ID# → COUNTY_NAME → TAX_RATE

PROPERTY_ID# → AREA → PRICE

By taking all the keys into account, the relation schema will only be in 1NF (just look at the general definitions of 2NF and 3NF). This is because there is a partial dependency on the secondary key COUNTY_NAME, LOT# listed here:

COUNTY_NAME → TAX_RATE

This violates 2NF.

15.23 Why do spurious tuples occur in the result of joining the EMP_PROJ1 and EMP_LOCS relations in Figure 15.5 (result shown in Figure 15.6)?

A tuple (e, l) in EMP_LOCS shows that we have an employee, named e, working on some project at location l. A tuple (s, p, h, pn, l) in EMP_PROJ1, shows that we have an employee working on project p at location l with social security number s. When these two are joined, the tuple (e, l) can be joined with the tuple (s, p, h, pn, l) where we have e, name of an employee, and s, social security number of a different employee. This results in spurious tuples, leaving us unhappy.

15.27 Consider a relation R(A, B, C, D, E) with the following dependencies:

$$\mathbf{AB \rightarrow C,\ CD \rightarrow E,\ DE \rightarrow B}$$

Is AB a candidate key of this relation? If not, is ABD? Explain your answer.

No, AB += {A,B,C}, which is a proper subset of {A,B,C,D,E}. Yes, ABD += {A,B,C,D,E}.

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

1. **What normal form is the relation in? Explain your answer.**
2. **Apply normalization until you cannot decompose the relations further.**

State the reasons behind each decomposition.

1. The key for this relation is Book_title, Authorname. Since no attributes are FFD on the key, this given relation is in 1NF. It is not in 2NF or 3NF.
2. (a) 2NF decomposition:
   Book0 (Book_title, Author_name)
   Book1 (Book_title, Publisher, Book_type, List_price)
   Book2 (Author_name, Author_affil)
   Any partial dependencies are eliminated by this decomposition.

   (b) 3NF decomposition:
   Book0 (Book_title, Author_name)
   Book1-1 (Book_title, Publisher, Book_type)
   Book1-2 (Book_type, List_price)
   Book2 (Author_name, Author_affil)
   Luckily, our decomposition eliminates any sort of transitive dependency on Listprice.

Chapter 16

16.18 Show that, if the matrix S resulting from Algorithm 16.3 does not have a row that is all a symbols, projecting S on the decomposition and joining it back will always produce at least one spurious tuple.

For every relation R i in our decomposition, the matrix S initially has one row with a symbols under the columns for the attributes. When we project S on each R i, the algorithm will produce one row consisting of all a symbols in each S(R i) since we never change an a symbol into a b symbol during the application of the algorithm. Joining all the a rows in each projection will give us at least one row of all a symbols. So, if S does not have a row that is all a, and then we apply this algorithm, our result will have at least one all a row. This means we have a spurious tuple.

16.19 Show that the relation schemas produced by Algorithm 16.5 are in BCNF.

The algorithm loop will only end after all relation schemas are in BCNF.

Chapter 17

17.27 Consider a disk with the following characteristics (these are not parameters of any particular disk unit): block size B = 512 bytes; interblock gap size G = 128 bytes; number of blocks per track = 20; number of tracks per surface = 400. A disk pack consists of 15 double-sided disks.

(a) What is the total capacity of a track, and what is its useful capacity (excluding interblock gaps)?

Total 20 * (512 + 128) = 12800 bytes = 12.8 Kbytes

Useful 20 * 512 = 10240 bytes = 10.24 Kbytes

(b) How many cylinders are there?

Tracks # tracks = # cylinders = 400

(c) What are the total capacity and the useful capacity of a cylinder?

Total 15 * 2 * 20 * (512 + 128) = 384000 bytes = 384 Kbytes

Useful 15 * 2 * 20 * 512 = 307200 bytes = 307.2 Kbytes

(d) What are the total capacity and the useful capacity of a disk pack?

Total 15 * 2 * 400 * 20 * (512 + 128) = 153600000 bytes = 153.6 Mbytes

Useful 15 * 2 * 400 * 20 * 512 = 122.88 Mbytes

(e) Suppose that the disk drive rotates the disk pack at a speed of 2400 rpm (revolutions per minute); what are the transfer rate (TR) in bytes/msec and the block transfer time (BTT) in msec? What is the average rotational delay (RD) in msec? What is the bulk transfer rate? (See Appendix B.)

Transfer rate TR = (total track size in bytes)/(time for one disk revolution in msec) TR = (12800) / ((60 * 1000) / (2400)) = (12800) / (25) = 512 bytes/msec

Block transfer time BTT = B / TR = 512 / 512 = 1 msec

Average rotational delay RD = (time for one disk revolution in msec) / 2 = 25 / 2 = 12.5 msec

Bulk transfer rate BTR = TR * (B / (B + G)) = 512 * (512/640) = 409.6 bytes/msec

(f) Suppose that the average seek time is 30 msec. How much time does it take (on the average) in msec to locate and transfer a single block, given its block address?

S + RD + BTT = 30 + 12.5 + 1 = 43.5 msec

(g) Calculate the average time it would take to transfer 20 random blocks, and compare this with the time it would take to transfer 20 consecutive blocks using double buffering to save seek time and rotational delay.

Random 20 * (S + RD + BTT) = 20 * 43.5 = 870 msec

Consecutive S + RD + 20 * BTT = 30 + 12.5 + (20 * 1) = 62.5 msec

17.37 Can you think of techniques other than an unordered overflow file that can be used to make insertions in an ordered file more efficient?

Keeping in mind the manner in which the overflow for static hash files is chained, using an overflow file in which the records are chained together is prudent. Any overflow records inserted in any block of the ordered file are linked together. Just like a linked list, a pointer to the first record in the chain is kept in the block of the main file. The list in our example can be kept ordered or unordered.

17.39 Can you think of techniques other than chaining to handle bucket overflow in external hashing?

Internal hashing has many techniwues which can handle bucket overflow. If a bucket is full, the record which should be inserted in that bucket may be placed in the next bucket if there is space, which is reminiscent of a technique called open addressing. Another way of approaching this problem is to simply consider using a whole overflow block for each and every bucket that becomes full.

Chapter 19

19.17 Can a nondense index be used in the implementation of an aggregate operator? Why or why not?

A nondense index contains entries for only some of the search values. A great example of a nondense index is a primary index because it includes an entry for every disk block of a data file as opposed to having one for every record. The nondense index would potentially be able to be used to compute the MIN function. This would be the case if the smallest key value in the data block corresponded to the keys in the index. Due to all the values not appearing in the index, MAX, AVG, SUM and COUNT would not be able to determined.

19.21 Extend the sort-merge join algorithm to implement the LEFT OUTER JOIN operation.

Before exteding the sort-merge join algorithmn, let's explain a LEFT OUTER JOIN operation. A LEFT OUTER JOIN of relations R and S would produce all of the rows from R and S that join. It would also produce the rows from R that *do not* join any row from S. In order to extend the sort-merge join algorithm, we only need to modify anything during the merge phase. In the first and second steps of the algorithm, the relations R and S would be soreted on their respective join column or columns. The third step would include them merging of the sorted R and S relations would be merged. If the R row and S row have the same value for the join column, their rows would be combined. Any given R row would also be placed in the LEFT OUTER JOIN result if no matching S row was found. The corresponding attributes from the S relation would simply be NULL values.

Chapter 22

22.22 Prove that strict two-phase locking guarantees strict schedules.

Due to the fact that any transaction that would want to read or write an item being read or written by transaction T must wait until T has committed, a strict schedule is satisfied due to this very condition.