CHAPTER 1: DATABASES AND DATABASE USERS

Answers to Selected Exercises

1.7 Give some informal queries and updates that you would expect to apply to the database shown in Figure 1.2.

Answer:

(a) (Query) List the names of all students majoring in Computer Science.

(b) (Query) What are the prerequisites of the Database course?.

(c) (Query) Retrieve the transcript of Smith. This is a list of <CourseName, SectionIdentifier, Semester, Year, Grade> for each course section that Smith has completed.

(d) (Update) Insert a new student in the database whose Name=Jackson, StudentNumber=23, Class=1 (freshman), and Major=MATH.

(e) (Update) Change the grade that Smith received in Intro to Computer Science section 119 to B.

1. 8 What is the difference between controlled and uncontrolled redundancy?

Answer:

Redundancy is when the same fact is stored multiple times in several places in a database. For example, in Figure 1.5(a) the fact that the name of the student with StudentNumber=8 is Brown is stored multiple times. Redundancy is controlled when the DBMS ensures that multiple copies of the same data are consistent; for example, if a new record with StudentNumber=8 is stored in the database of Figure 1.5(a), the DBMS will ensure that StudentName=Smith in that record. If the DBMS has no control over this, we have uncontrolled redundancy.

1. 9 Give all the relationships among the records of the database shown in Figure 1.2.

Answer:

(a) Each SECTION record is related to a COURSE record.

(b) Each GRADE_REPORT record is related to one STUDENT record and one SECTION record.

(c) Each PREREQUISITE record relates two COURSE records: one in the role of a course and the other in the role of a prerequisite to that course.

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

Answer:

(a) A view that groups all the students who took each section and gives each student's grade. This may be useful for printing the grade report for each section for the university administration's use.

(b) A view that gives the number of courses taken and the GPA (grade point average) for each student. This may be used to determine honors students.

1.11 Give some examples of integrity constraints that you think should hold on the database shown in Figure 1.2.

Answer:

We give a few constraints expressed in English. Following each constraint, we give its type in the relational database terminology that will be covered in Chapter 6, for reference purposes.

(a) The StudentNumber should be unique for each STUDENT record (key constraint).

(b) The CourseNumber should be unique for each COURSE record (key constraint).

(c) A value of CourseNumber in a SECTION record must also exist in some COURSE record (referential integrity constraint).

(d) A value of StudentNumber in a GRADE_REPORT record must also exist in some STUDENT record (referential integrity constraint).

(e) The value of Grade in a GRADE_REPORT record must be one of the values in the set {A, B, C, D, F, I, U, S} (domain constraint).

(f) Every record in COURSE must have a value for CourseNumber (entity integrity constraint).

(g) A STUDENT record cannot have a value of Class=2 (sophomore) unless the student has completed a number of sections whose total course CreditHours is greater that 24 credits (general semantic integrity constraint).

CHAPTER 2: DATABASE SYSTEM CONCEPTS AND ARCHITECTURE

Answers to Selected Exercises

2.11 Think of different users for the database of Figure 1.2. What type of applications would each user need? To which user category would each belong and what type of interface would they need?

Answer:

(a) Registration Office User: They can enter data that reflect the registration of students in sections of courses, and later enter the grades of the students. Applications can include:
- Register a student in a section of a course
- Check whether a student who is registered in a course has the appropriate prerequisite courses
- Drop a student from a section of a course
- Add a student to a section of a course
- Enter the student grades for a section

Application programmers can write a number of canned transactions for the registration office end-users, providing them with either forms and menus, or with a parametric interface.

(b) Admissions Office User: The main application is to enter newly accepted students into the database. Can use the same type of interfaces as (a).

(c) Transcripts Office User: The main application is to print student transcripts. Application programmers can write a canned transaction using a report generator utility to print the transcript of a student in a prescribed format. The particular student can be identified by name or social security number. Another application would be to generate grade slips at the end of each semester for all students who have completed courses during that semester. Again, this application could be programmed using a report generator utility.

2.12 - No Solution provided

CHAPTER 3: DATA MODELING USING THE ENTITY-RELATIONSHIP MODEL

Answers to Selected Exercises

3.16 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, permanent address and phone, birthdate, 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 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, and college. Both name and code have unique values for each department.

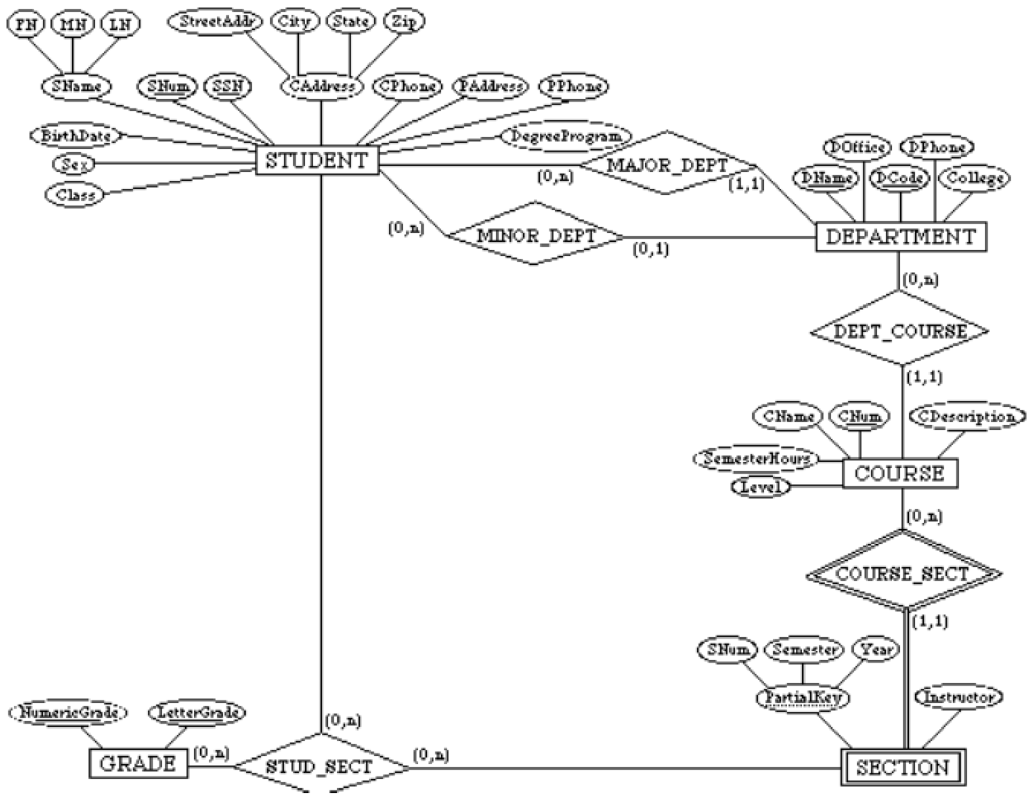
(c) Each course has a course name, description, course number, number of semester hours, level, and offering department. The value of course number is unique for each course.

(d) Each section has an instructor, semester, year, course, and section number. The section number distinguishes different 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, 4 for F, D, C, B, A, respectively).

Design an ER schema for this application, and draw an ER diagram for that 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.

Answer:

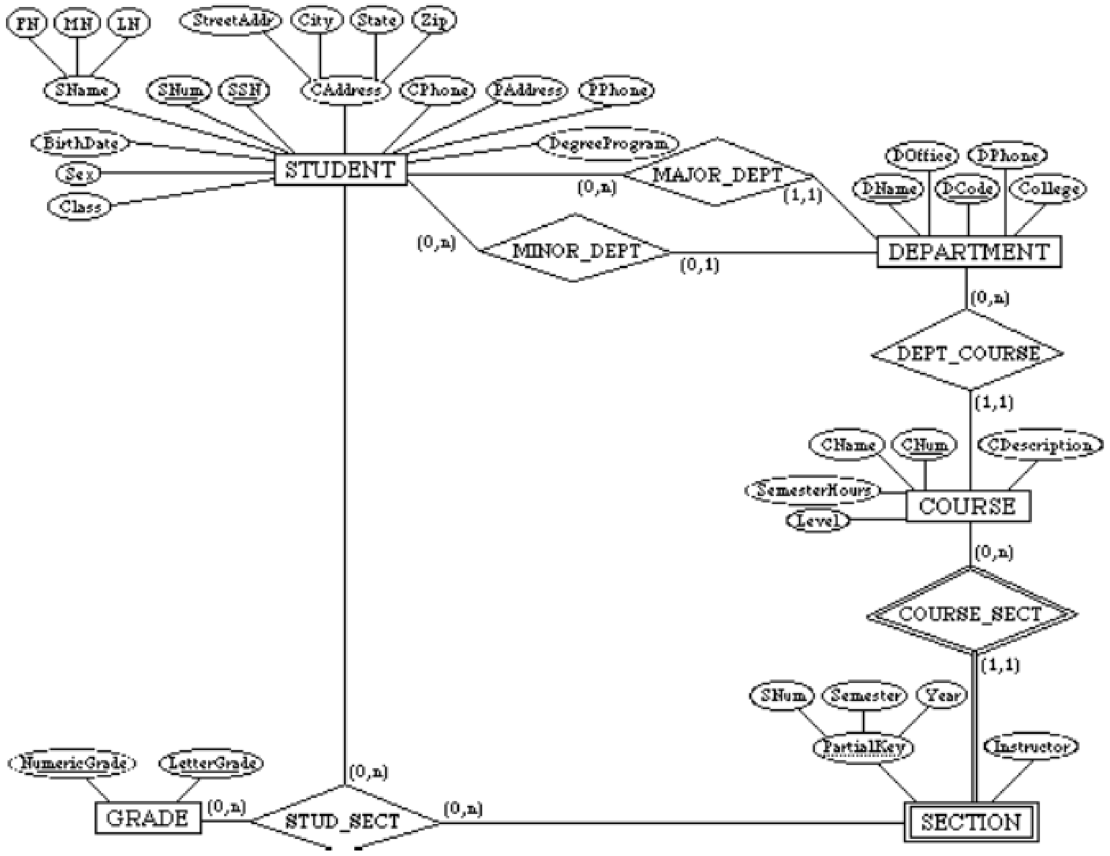


ER Schema diagram for exercise 3.16:

3.17 Composite and multi-valued attributes can be nested to any number of levels. Suppose we want to design an attribute for a STUDENT entity type to keep track of previous college education. Such an attribute will have one entry for each college previously attended, and this entry is composed of: college name, start and end dates, degree entries (degrees awarded at that college, if any), and transcript entries (courses completed at that college, if any). Each degree entry is formed of degree name and the month and year it was awarded, and each transcript entry is formed of a course name, semester, year, and grade. Design an attribute to hold this information. Use the conventions of Figure 3.5.

Answer:

{ PreviousEducation (CollegeName, StartDate, EndDate,
{ Degree (DegreeName, Month, Year) },
{ Transcript (CourseName, Semester, Year, Grade) }) }

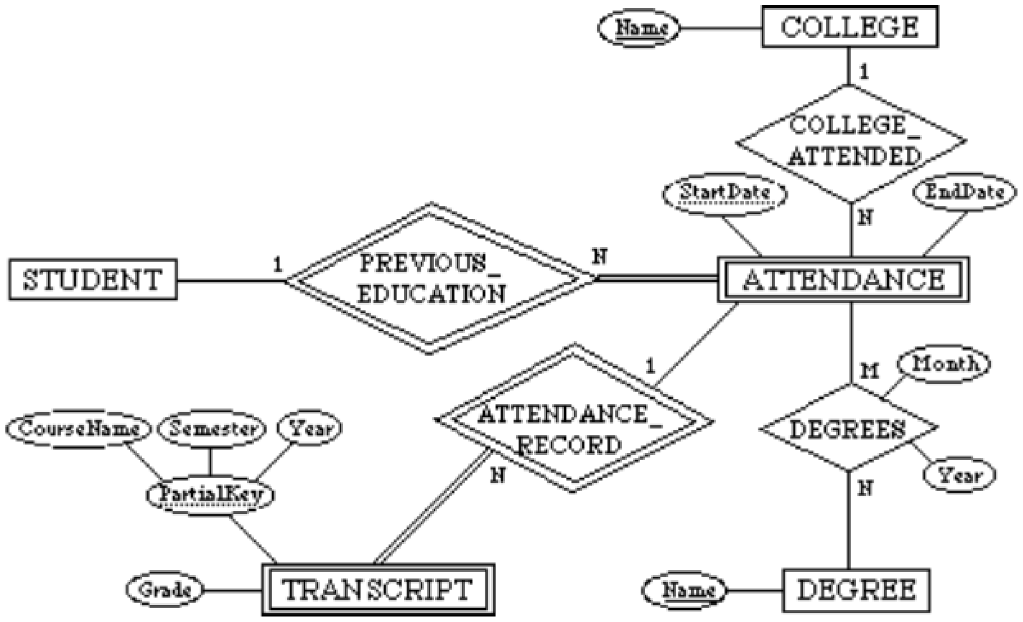


ER Schema Diagram for Exercise 3.16

3.18 Show an alternative design for the attribute described in Exercise 3.17 that uses only entity types (including weak entity types if needed) and relationship types.

Answer:

This example illustrates a perceived weakness of the ER model, which is: how does the database designer decide what to model as an entity type and what to model as a relationship type. In our solution, we created a weak entity type ATTENDANCE; each (weak) entity in ATTENDANCE represents a period in which a STUDENT attended a particular COLLEGE, and is identified by the STUDENT and the StartDate of the period. Hence, the StartDate attribute is the partial key of ATTENDANCE. Each ATTENDANCE entity is related to one COLLEGE and zero or more DEGREEs (the degrees awarded during that attendance period). The TRANSCRIPT of the STUDENT during each attendance period is modeled as a weak entity type, which gives the records of the student during the attendance period. Each (weak) entity in TRANSCRIPT gives the record of the sudent in one course during the attendance period, as shown in the ER diagram below. Other ER schema designs are also possible for this problem.



3.19 Consider the ER diagram of Figure 3.17, which shows a simplified schema for an airline reservations system. Extract from the ER diagram the requirements and constraints that resulted in this schema. Try to be as precise as possible in your requirements and constraints specification.

Answer:

(1) The database represents each AIRPORT, keeping its unique AirportCode, the AIRPORT Name, and the City and State in which the AIRPORT is located.

(2) Each airline FLIGHT has a unique number, the Airline for the FLIGHT, and the Weekdays on which the FLIGHT is scheduled (for example, every day of the week except Sunday can be coded as X7).

(3) A FLIGHT is composed of one or more FLIGHT LEGs (for example, flight number CO1223 from New York to Los Angeles may have two FLIGHT LEGs: leg 1 from New York to Houston and leg 2 from Houston to Los Angeles). Each FLIGHT LEG has a DEPARTURE AIRPORT and Scheduled Departure Time, and an ARRIVAL AIRPORT and Scheduled Arrival Time.

(4) A LEG INSTANCE is an instance of a FLIGHT LEG on a specific Date (for example, CO1223 leg 1 on July 30, 1989). The actual Departure and Arrival AIRPORTs and Times are recorded for each flight leg after the flight leg has been concluded. The Number of available seats and the AIRPLANE used in the LEG INSTANCE are also kept.

(5) The customer RESERVATIONs on each LEG INSTANCE include the Customer Name,

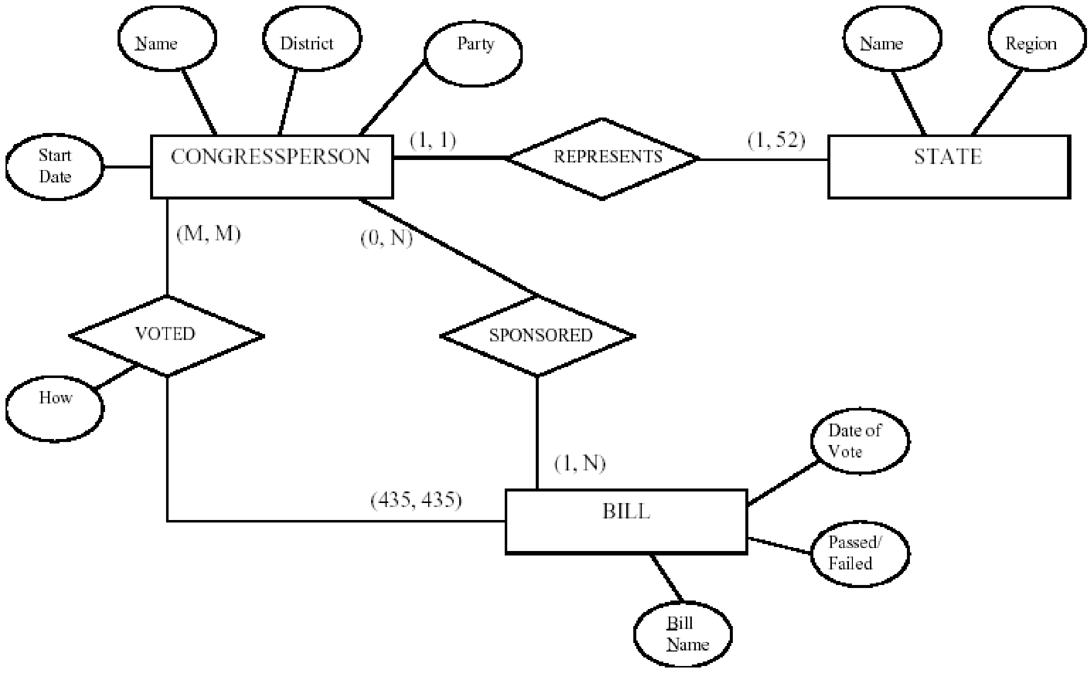
Phone, and Seat Number(s) for each reservation.

(6) Information on AIRPLANEs and AIRPLANE TYPEs are also kept. For each AIRPLANE TYPE (for example, DC-10), the TypeName, manufacturing Company, and Maximum Number of Seats are kept. The AIRPORTs in which planes of this type CAN LAND are kept in the database. For each AIRPLANE, the AirplaneId, Total number of seats, and TYPE are kept.

3.20 No solution provided

3.21.
Additional information:
- There are 435 congresspersons in the U.S. House of Representatives.
- States have between one (AK, DE, MT, ND, SD, VT, and WY) and 52 (CA) representatives.
- M represents number of bills during the 2-year session.

The resulting ER Diagram is shown in Figure A.

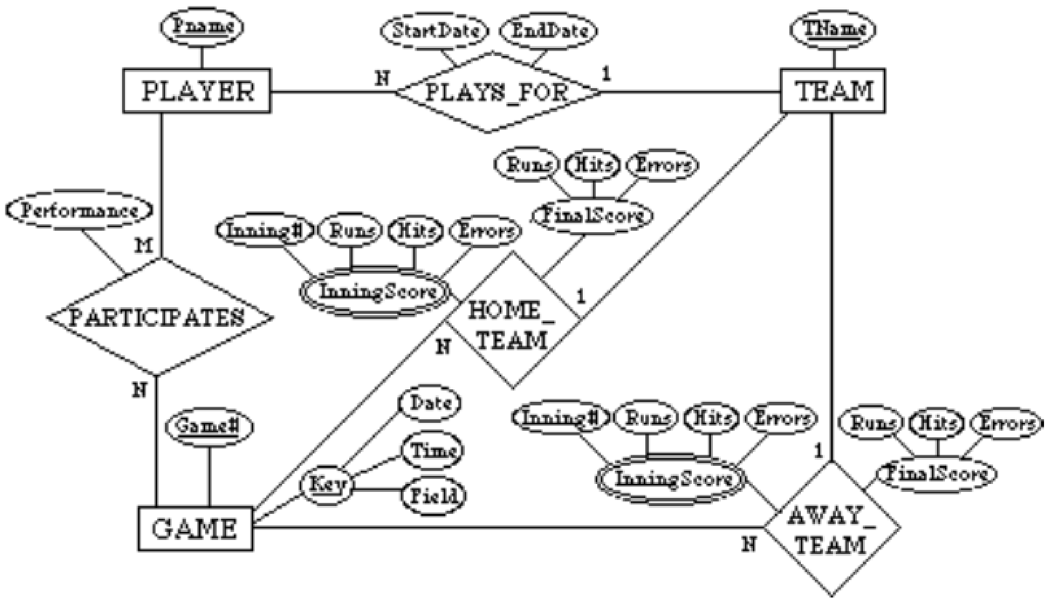


3.22 A database is being constructed to keep track of the teams and games of a sports league. A team has a number of players, not all of whom participate in each game. It is desired to keep track of the players participating in each game for each team, the positions they played in that game, and the result of the game. Try to design an ER schema diagram for this application, stating any assumptions you make. Choose your favorite sport (soccer, football, baseball, ...).

Answer:

The following design may be used for a baseball league. Here, we assumed that each game in the schedule is identified by a unique Game#, and a game is also identified uniquely by the combination of Date, starting Time, and Field where it is played. The Performance attribute of PARTICIPATE is used to store information on the individual performance of each player in a game. This attribute can be designed to keep the information needed for statistics, and may be quite complex. One possible design for the Performance attribute may be the following (using the notation of Figure 3.8):

Performance({Hitting(AtBat#, Inning#, HitType, Runs, RunsBattedIn, StolenBases)},
{Pitching(Inning#, Hits, Runs, EarnedRuns, StrikeOuts, Walks, Outs,
Balks, WildPitches)},
{Defense(Inning#, {FieldingRecord(Position, PutOuts, Assists, Errors)})})
Here, performance is a composite attribute made up of three multivalued components: Hitting, Pitching, and Defense. Hitting has a value for each AtBat of a player, and records the HitType (suitable coded; for example, 1 for single, 2 for double, 3 for triple, 4 for home run, 0 for walk, -1 for strikeout, -2 for fly out, ...) and other information concerning the AtBat. Pitching has a value for each inning during which the player pitched. Defense has a value for each inning a player played a fielding position. We can have a less detailed or a more detailed design for the performance of a player in each game, depending on how much information we need to keep in the database. Suitable variations of the ER diagram shown below can be used for other sports.



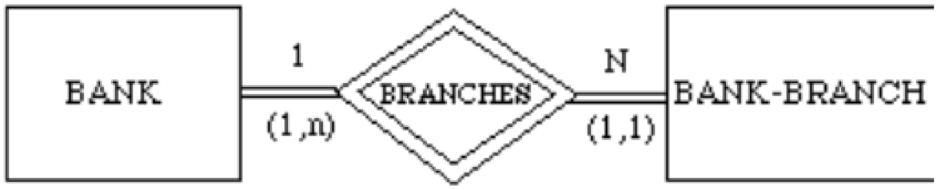
3.23 Consider the ER diagram shown in Figure 3.18 for part of a BANK database. Each bank can have multiple branches, and each branch can have multiple accounts and loans.

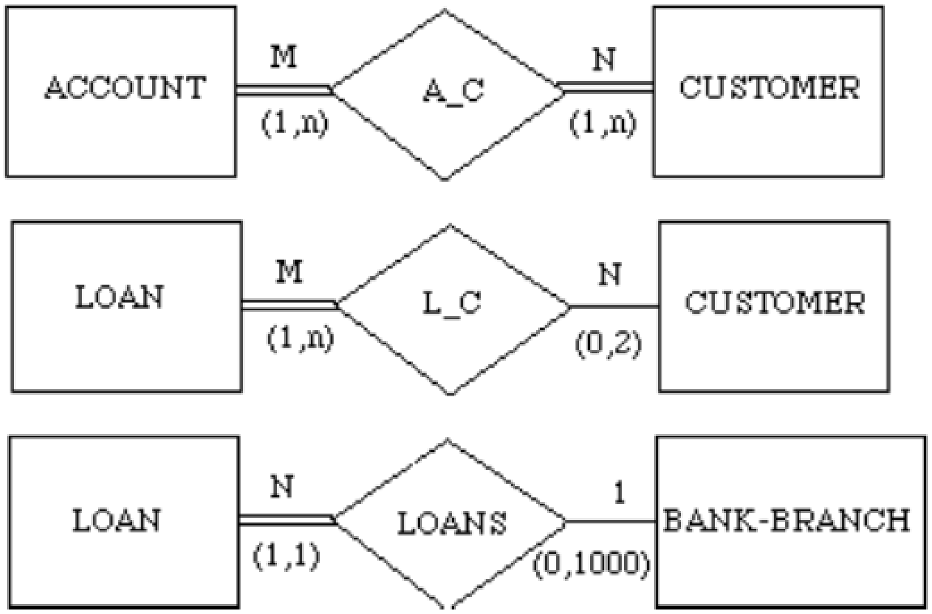
(a) List the (nonweak) entity types in the ER diagram.

(b) Is there a weak entity type? If so, give its name, its partial key, and its identifying relationship.

(c) What constraints do the partial key and the identifying relationship of the weak entity type specify in this diagram?

(d) List the names of all relationship types, and specify the (min,max) constraint on each participation of an entity type in a relationship type. Justify your choices.





(e) List concisely the user requirements that led to this ER schema design.

(f) Suppose that every customer must have at least one account but is restricted to at most two loans at a time, and that a bank branch cannot have more than 1000 loans. How does this show up on the (min,max) constraints?

Answer:

(a) Entity types: BANK, ACCOUNT, CUSTOMER, LOAN

(b) Weak entity type: BANK-BRANCH. Partial key: BranchNo.
Identifying relationship: BRANCHES.

(c) The partial key BranchNo in BANK-BRANCH specifies that the same BranchNo value may occur under different BANKs. The identifying relationship BRANCHES specifies that BranchNo values are uniquely assigned for those BANK-BRANCH entities that are related to the same BANK entity. Hence, the combination of BANK Code and BranchNo together constitute a full identifier for a BANK-BRANCH.

(d) Relationship Types: BRANCHES, ACCTS, LOANS, A-C, L-C. The (min, max) constraints are shown below.

(e) The requirements may be stated as follows: Each BANK has a unique Code, as well as a Name and Address. Each BANK is related to one or more BANK-BRANCHes, and the BranhNo is unique among each set of BANK-BRANCHes that are related to the same BANK. Each BANK-BRANCH has an Address. Each BANK-BRANCH has zero or more LOANS and zero or more ACCTS. Each ACCOUNT has an AcctNo (unique), Balance, and Type and is related to exactly one BANK-BRANCH and to at least one CUSTOMER. Each LOAN has a LoanNo (unique), Amount, and Type and is related to exactly one BANK-BRANCH and to at least one CUSTOMER. Each CUSTOMER has an SSN (unique), Name, Phone, and Address, and is related to zero or more ACCOUNTs and to zero or more LOANs.

(f) The (min, max) constraints would be changed as follows:

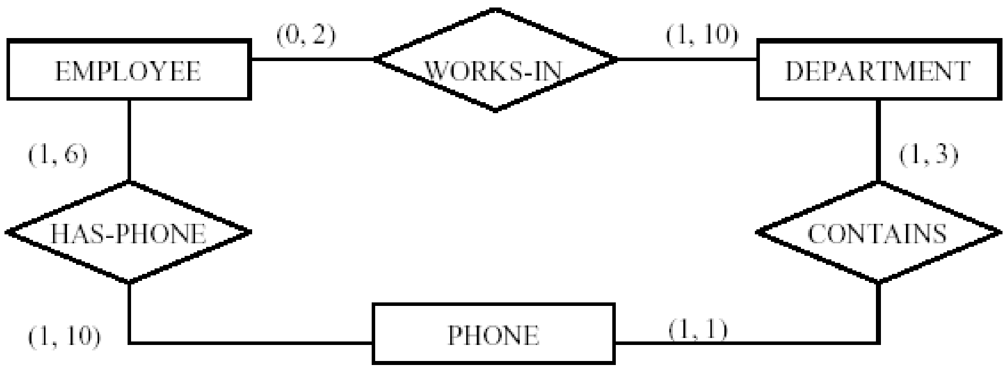
3.24 Consider the ER diagram in Figure 3.19. Assume that an employee may work in up to two departments or may not be assigned o any department. Assume that each department must have one and may have up to three phone numbers. Supply (min, max) constraints on this diagram. State clearly any additional assumptions you make. Under what conditions would the relationship HAS_PHONE be redundant in this example?

Answer:

Assuming the following additional assumptions:
- Each department can have anywhere between 1 and 10 employees.
- Each phone is used by one, and only one, department.
- Each phone is assigned to at least one, and may be assigned to up to 10 employees.
- Each employee is assigned at least one, but no more than 6 phones.

The resulting ER Diagram will have the (min, max) constraints shown in Figure A.

Figure A



Relationship HAS-PHONE would be redundant under the following conditions:
- Each employee is assigned all of the phones of each department that he/she works in.
- An employee cannot have any other phones outside the departments he/she works is.
EMPLOYEE
PHONE
DEPARTMENT
HAS-PHONE CONTAINS
WORKS-IN
(0, 2)
(1, 3)
(1, 10)
(1, 10) (1, 1)
(1, 6)

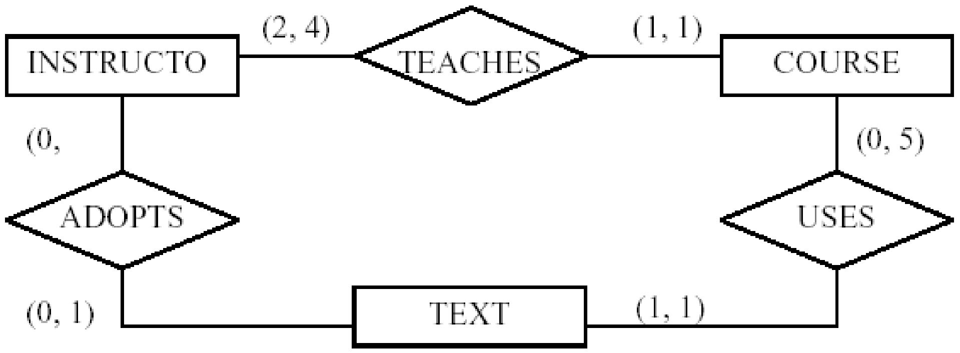
3.25. Consider the ER diagram in Figure 3.20. Assume that a course may or may not use a textbook, but that a text by definition is a book that is used in some course. A course may not use more than five books. Instructors teach from two to four courses. Supply (min, max) constraints on this diagram. State clearly any additional assumptions you make. If we add the relationship ADOPTS between INSTRUCTOR and TEXT, what (min, max) constraints would you put on it? Why?

Answer:

Assuming the following additional assumptions:
- Each course is taught by exactly one instructor.
- Each textbook is used by one and only one course.
- An instructor does not have to adopt a textbook for all courses.
- If a text exists:
- ___it is used in some course,
- ___hence it is adopted by some instructor who teaches that course.
- An instructor is considered to adopt a text if it is used in some course taught
- by that instructor.

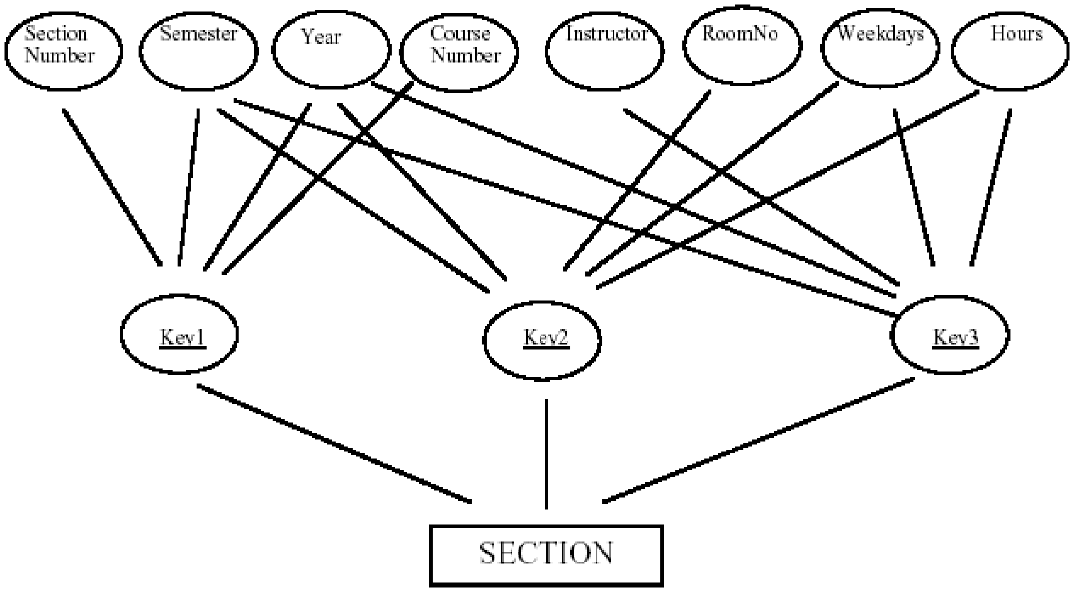
The resulting ER Diagram will have the (min, max) constraints shown in Figure B.

Figure B



3.26 Consider an entity type SECTION in a UNIVERSITY database, which describes the section offerings of courses. The attributes of SECTION are SectionNumber, Semester, Year, CourseNumber, Instructor, RoomNo (where section is taught), Building (where section is taught), Weekdays (domain is the possible combinations of weekdays in which a section can be offered {MWF, MW, TT, etc.}). Assume tat SectionNumber is unique for each course within a particular semester/year combination (that is, if a course if offered multiple times during a particular semester, its section offerings are numbered 1, 2, 3, etc.). There are several composite keys for SECTION, and some attribute sare components of more than one key. Identify three composite keys, and show how they can be represented in an ER schema diagram.

Answer:

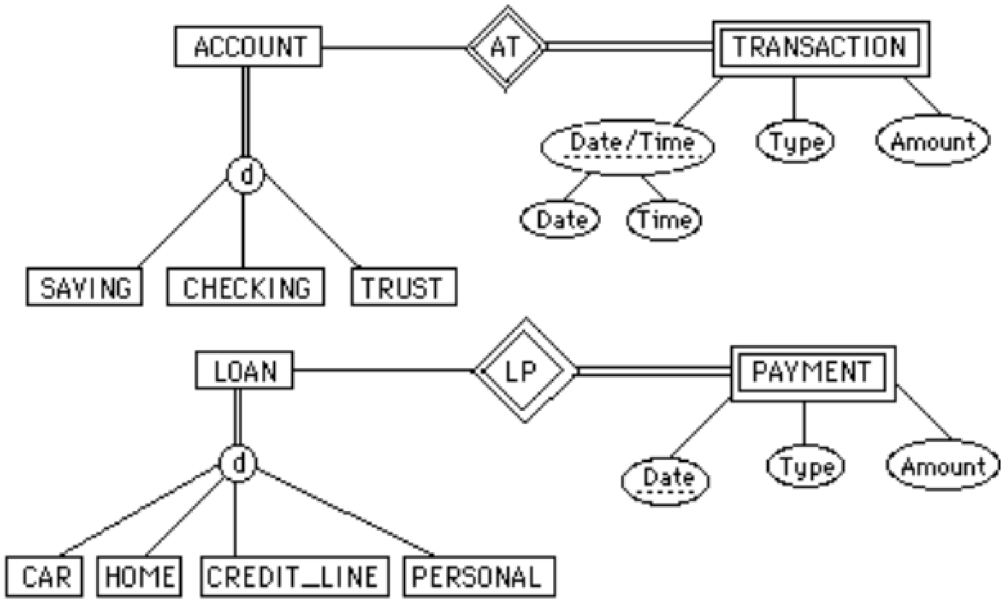


CHAPTER 4: ENHANCED ENTITY-RELATIONSHIP AND UML MODELING

Answers to Selected Exercises

4.18 Consider the BANK ER schema of Figure 3.18, and suppose that is is necessary to keep track of different types of ACCOUNTS (SAVINGS_ACCTS, CHECKING_ACCTS, ...) and LOANS (CAR_LOANS, HOME_LOANS, ...). Suppose that is 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, time, ... Modify the BANK schema, using ER and EER concepts of specialization and generalization. State any assumptions you make about the additional requirements.

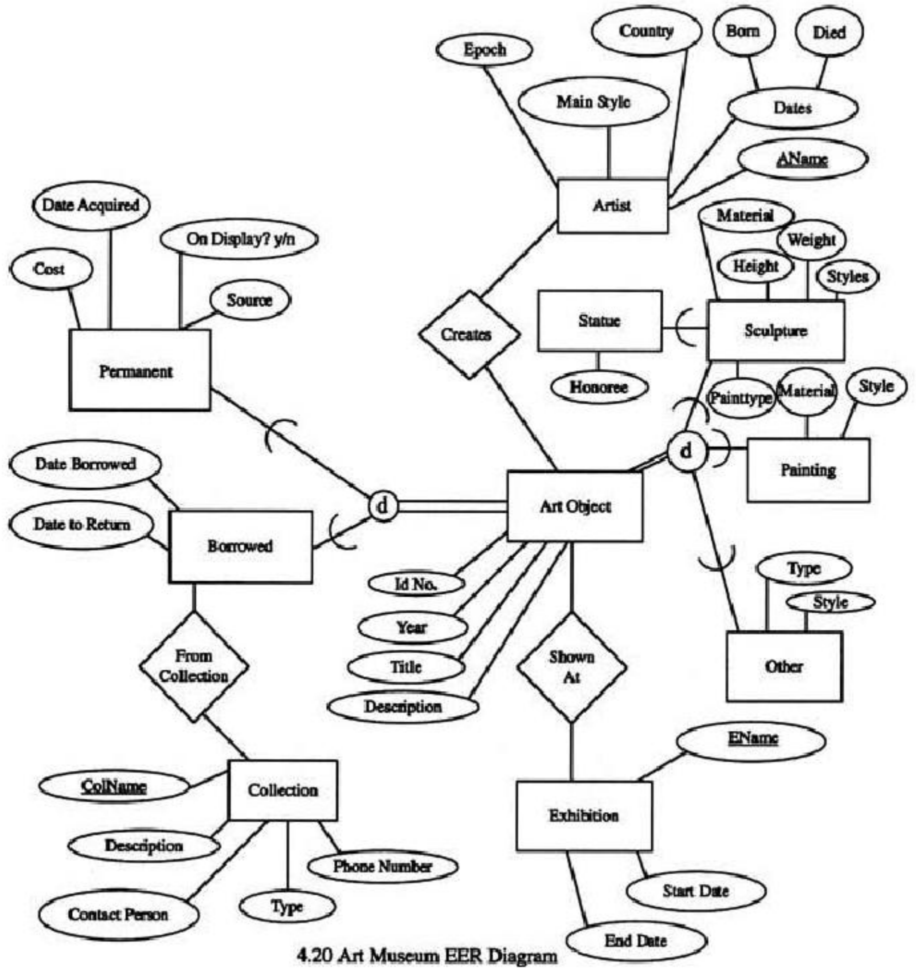
Answer:



4.19 Solution Pending

4.20 Identify all the important concepts represented in the library database case study described here. In particular, identify the abstraction of classification (entity types and relationship types), aggregation, identification, and specialization/generalization. Specify (min, max) cardinality constraints whenever possible. List details that will affect the eventual design but have no bearing on the conceptual design. List the semantic separately. Draw an EER diagram of the library database.

Answer:



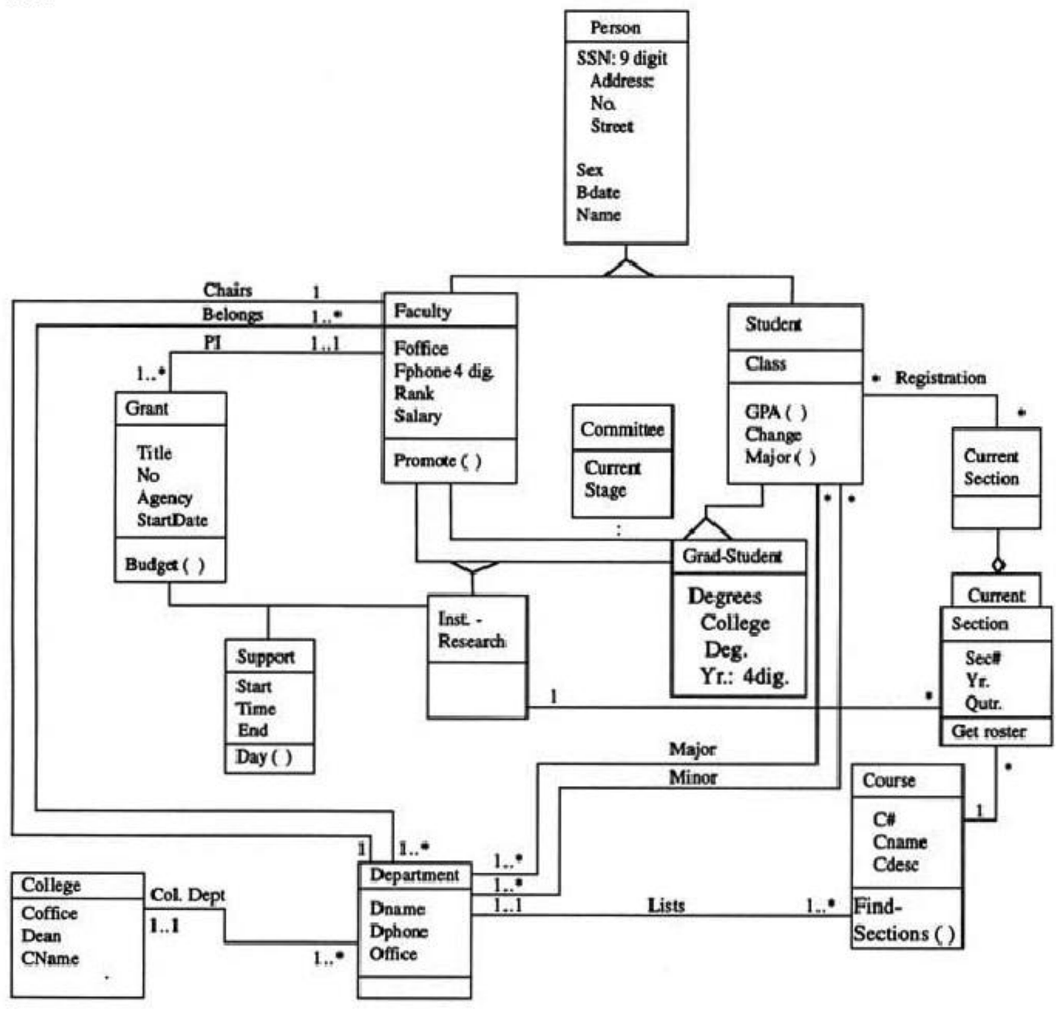
4.20 Art Museum EER Diagram

4.21 Solution Pending

4.22 Figure 4.15 shows an example of an EER diagram for a small private airport database that is used to keep track of airplanes, their owners, airport employees, and pilots. From the requirements for this database, the following information was collected: Each airplane has a registration number [Reg#], is of a particular plane type [of_type], and is stored in a particular hangar [stored_in]. Each plane_type has a model number [Model], a capacity [Capacity], and a weight [Weight]. Each hangar has a number [Number], a capacity [Capacity], and a location [Location]. The database also keeps track of the owners of each plane [owns] and the employees who have maintained the plane [maintain]. Each relationship instance in owns relates an airplane to an owner and includes the purchase date [Pdate]. Each relationship instance in maintain relates to an employee to a service record [service]. Each plane undergoes service many times; hence, it is related by [plane_service] o a number of service records. A service record includes as attributes the date of maintenance [Date], the number of hours spent on the work [Hours], and the type of work done [Workcode]. We use a weak entity type [service] to represent airplane service, because the airplane registration number is used to identify a service record. An owner is either a person or a corporation. Hence, we use a union type (category) [owner] that is a subset of the union of corporation [Corporation] and person [Person] entity types. Both pilots {Pilot] and employees [Employee] are subclasses of person. Each pilot has specific attributes license number [Lic_Num] and restrictions [Restr]; each employee has specific attributes salary [Salary] and shift {Shift]. All person entities in the database have data kept on their social security number [Ssn], name

[Name], address [Address], and telephone number [Phone]. For corporation entities, the data kept includes name [Name], address [Address], and telephone number [Phone]. The database also keeps track of the types of planes each pilot is authorized to fly [Flies] and the types of planes each employee can do maintenance work on [Works_on]. Show how the small airport EER schema of Figure 4.15 may be represented in UML notation. (Note: We have not discussed how to represent categories (union types) in UML, so you do not have to map the categories in this and the following question.)

Answer:



CHAPTER 5: THE RELATIONAL DATA MODEL AND RELATIONAL DATABASE CONSTRAINTS

Answers to Selected Exercises

5.10 Suppose each of the following update operations is applied directly to the database of Figure 5.6. Discuss all integrity constraints violated by each operation, if any, and the different ways of enforcing these constraints:

(a) Insert < 'Robert', 'F', 'Scott', '943775543', '21-JUN-42', '2365 Newcastle Rd, Bellaire, TX', M, 58000, '888665555', 1 > into EMPLOYEE.

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

(c) Insert < 'Production', 4, '943775543', '01-OCT-88' > into DEPARTMENT.

(d) Insert < '677678989', null, '40.0' > into WORKS_ON.
(e) Insert < '453453453', 'John', M, '12-DEC-60', 'SPOUSE' > into DEPENDENT.

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

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

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

(i) Modify the MGRSSN and MGRSTARTDATE of the DEPARTMENT tuple with DNUMBER=5 to '123456789' and '01-OCT-88', respectively.

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

(k) Modify the HOURS attribute of the WORKS_ON tuple with ESSN= '999887777' and PNO= 10 to '5.0'.

Answers:

(a) No constraint violations.

(b) Violates referential integrity because DNUM=2 and there is no tuple in the DEPARTMENT relation with DNUMBER=2. We may enforce the constraint by: (i) rejecting the insertion of the new PROJECT tuple, (ii) changing the value of DNUM in the new PROJECT tuple to an existing DNUMBER value in the DEPARTMENT relation, or (iii) inserting a new DEPARTMENT tuple with DNUMBER=2.

(c) Violates both the key constraint and referential integrity. Violates the key constraint because there already exists a DEPARTMENT tuple with DNUMBER=4. We may enforce this constraint by: (i) rejecting the insertion, or (ii) changing the value of DNUMBER in the new DEPARTMENT tuple to a value that does not violate the key constraint. Violates referential integrity because MGRSSN='943775543' and there is no tuple in the EMPLOYEE relation with SSN='943775543'. We may enforce the constraint by: (i) rejecting the insertion, (ii) changing the value of MGRSSN to an existing SSN value in EMPLOYEE, or (iii) inserting a new EMPLOYEE tuple with SSN='943775543'.

(d) Violates both the entity integrity and referential integrity. Violates entity integrity because PNO, which is part of the primary key of WORKS_ON, is null. We may enforce this constraint by: (i) rejecting the insertion, or (ii) changing the value of PNO in the new WORKS_ON tuple to a value of PNUMBER that exists in the PROJECT relation. Violates referential integrity because ESSN='677678989' and there is no tuple in the EMPLOYEE relation with SSN='677678989'. We may enforce the constraint by: (i) rejecting the insertion, (ii) changing the value of ESSN to an existing SSN value in EMPLOYEE, or (iii) inserting a new EMPLOYEE tuple with SSN='677678989'.

(e) No constraint violations.

(f) No constraint violations.

(g) Violates referential integrity because several tuples exist in the WORKS_ON, DEPENDENT, DEPARTMENT, and EMPLOYEE relations that reference the tuple being deleted from EMPLOYEE. We may enforce the constraint by: (i) rejecting the deletion, or (ii) 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) Violates referential integrity because two tuples exist in the WORKS_ON relations that reference the tuple being deleted from PROJECT. We may enforce the constraint by: (i) rejecting the deletion, or (ii) deleting the tuples in the WORKS_ON relation whose value for PNO=1 (the value for the primary key PNUMBER for the tuple being deleted from PROJECT).

(i) No constraint violations.

(j) Violates referential integrity because the new value of SUPERSSN='943775543' and there is no tuple in the EMPLOYEE relation with SSN='943775543'. We may enforce the constraint by: (i) rejecting the deletion, or (ii) inserting a new EMPLOYEE tuple with SSN='943775543'.

(k) No constraint violations.

5.11 Consider the AIRLINE relational database schema shown in Figure 5.8, which describes a database for airline flight information. Each FLIGHT is identified by a flight NUMBER, and consists of one or more FLIGHT_LEGs with LEG_NUMBERs 1, 2, 3, etc. Each leg has scheduled arrival and departure times and airports, and has many LEG_INSTANCEs--one for each DATE on which the flight travels. FARES are kept for each flight. For each leg instance, SEAT_RESERVATIONs are kept, as is the AIRPLANE used in the leg, and the actual arrival and departure times and airports. An AIRPLANE is identified by an AIRPLANE_ID, and is of a particular AIRPLANE_TYPE. CAN_LAND relates AIRPLANE_TYPEs to the AIRPORTs in which they can land. An AIRPORT is identified by an AIRPORT_CODE. Consider an update for the AIRLINE database to enter a reservation on a particular flight or flight leg on a given date.

(a) Give the operations for this update.

(b) What types of constraints would you expect to check?

(c) Which of these constraints are key, entity integrity, and referential integrity constraints and which are not?

(d) Specify all the referential integrity constraints on Figure 5.8.

Answers:

(a) One possible set of operations for the following update is the following:
INSERT <FNO,LNO,DT,SEAT_NO,CUST_NAME,CUST_PHONE> into
SEAT_RESERVATION; MODIFY the LEG_INSTANCE tuple with the condition:
(FLIGHT_NUMBER=FNO AND LEG_NUMBER=LNO AND DATE=DT) by setting
NUMBER_OF_AVAILABLE_SEATS = NUMBER_OF_AVAILABLE_SEATS - 1; These
operations should be repeated for each LEG of the flight on which a reservation is
made. This assumes that the reservation has only one seat. More complex operations will
be needed for a more realistic reservation that may reserve several seats at once.

(b) We would check that NUMBER_OF_AVAILABLE_SEATS on each LEG_INSTANCE of
the flight is greater than 1 before doing any reservation (unless overbooking is permitted),
and that the SEAT_NUMBER being reserved in SEAT_RESERVATION is available.

(c) The INSERT operation into SEAT_RESERVATION will check all the key, entity integrity,
and referential integrity constraints for the relation. The check that
NUMBER_OF_AVAILABLE_SEATS on each LEG_INSTANCE of the flight is greater than 1
does not fall into any of the above types of constraints (it is a general semantic integrity
constraint).

(d) We will write a referential integrity constraint as R.A --> S (or R.(X) --> T)
whenever attribute A (or the set of attributes X) of relation R form a foreign key that
references the primary key of relation S (or T). FLIGHT_LEG.FLIGHT_NUMBER --> FLIGHT
FLIGHT_LEG.DEPARTURE_AIRPORT_CODE --> AIRPORT
FLIGHT_LEG.ARRIVAL_AIRPORT_CODE --> AIRPORT
LEG_INSTANCE.(FLIGHT_NUMBER,LEG_NUMBER) --> FLIGHT_LEG
LEG_INSTANCE.DEPARTURE_AIRPORT_CODE --> AIRPORT
LEG_INSTANCE.ARRIVAL_AIRPORT_CODE --> AIRPORT
LEG_INSTANCE.AIRPLANE_ID --> AIRPLANE
FARES.FLIGHT_NUMBER --> FLIGHT
CAN_LAND.AIRPLANE_TYPE_NAME --> AIRPLANE_TYPE
CAN_LAND.AIRPORT_CODE --> AIRPORT
AIRPLANE.AIRPLANE_TYPE --> AIRPLANE_TYPE
SEAT_RESERVATION.(FLIGHT_NUMBER,LEG_NUMBER,DATE) --> LEG_INSTANCE

5.12 Consider the relation CLASS(Course#, Univ_Section#, InstructorName, Semester,
BuildingCode, Room#, TimePeriod, Weekdays, CreditHours). This represents classes taught
in a university with unique Univ_Section#. Give what you think should be various candidate
keys and write in your own words under what constraints each candidate key would be valid.

Answer:

Possible candidate keys include the following (Note: We assume that the values of the
Semester attribute include the year; for example "Spring/94" or "Fall/93" could be
values for Semester):
1. {Semester, BuildingCode, Room#, TimePeriod, Weekdays} if the same room cannot be
used at the same time by more than one course during a particular semester.
2. {Univ_Section#} if it is unique across all semesters.
3. {InstructorName, Semester} if an instructor can teach at most one course during each

semester.
4. If Univ_Section# is not unique, which is the case in many universities, we have to examine the rules that the university uses for section numbering. For example, if the sections of a particular course during a particular semester are numbered 1, 2, 3, ..., then a candidate key would be {Course#, Univ_Section#, Semester}. If, on the other hand, all sections (of any course) have unique numbers during a particular semester only, then the candidate key would be {Univ_Section#, Semester}.

5.13 Consider the following six relations for an order-processing database application in a company:

CUSTOMER (<u>Cust#</u>, Cname, City)
ORDER (<u>Order#</u>, Odate, Cust#, Ord_Amt)
ORDER_ITEM (<u>Order#</u>, <u>Item#</u>, Qty)
ITEM (<u>Item#</u>, Unit_price)
SHIPMENT (<u>Order#</u>, <u>Warehouse#</u>, Ship_date)
WAREHOUSE (<u>Warehouse#</u>, City)

Here, Ord_Amt refers to total dollar amount of an order; Odate is the date the order was placed; Ship_date is the date an order is shipped from the warehouse. Assume that an order can be shipped from several warehouses. Specify foreign keys for this schema, stating any assumptions you make.

Answer:

Strictly speaking, a foreign key is a *set* of attributes, but when that set contains only one attribute, then that attribute itself is often informally called a foreign key. The schema of this question has the following five foreign keys:
1. the attribute Cust# of relation ORDER that references relation CUSTOMER,
2. the attribute Order# of relation ORDER_ITEM that references relation ORDER,
3. the attribute Item# of relation ORDER_ITEM that references relation ITEM,
4. the attribute Order# of relation SHIPMENT that references relation ORDER, and
5. the attribute Warehouse# of relation SHIPMENT that references relation WAREHOUSE.
We now give the queries in relational algebra:

a. SHIP_W2 _ _ $_{\text{Warehouse\# = 'W2'}}$ (SHIPMENT)
RESULT _ $\pi_{\text{Order\#, Ship\_date}}$ (SHIP_W2)

b. ORDERS_JL _ ORDER * _ $_{\text{Cname = 'Jose Lopez'}}$ (CUSTOMER)
RESULT _ $\pi_{\text{Order\#, Warehouse\#}}$ (ORDERS_JL * SHIPMENT)

c. BY_CUSTNUM _ $_{\text{Cust\#}}$ $\mathcal{F}_{\text{COUNT Order\#, AVERAGE Ord\_Amt}}$ (ORDER)
RESULT (CUSTNAME, #OFORDERS, AVG_ORDER_AMT) _
$\pi_{\text{Cname, COUNT\_Order\#, AVERAGE\_Ord\_Amt}}$ (BY_CUSTNUM * CUSTOMER)

d. TIMELY_SHIPPED _ _ $_{\text{Ship\_date} \leq \text{Odate} + 30}$ (ORDER * SHIPMENT)
RESULT _ $\pi_{\text{Order\#}}$ (ORDER) − $\pi_{\text{Order\#}}$ (TIMELY_SHIPPED)

The above query lists all orders for which no “timely” shipment was made, including orders for which no shipment was ever made. It is instructive to compare the above query with the one below that lists those orders for which at least one “late” shipment was made.

LATE_SHIPPED _ _ $_{\text{Ship\_date} > \text{Odate} + 30}$ (ORDER * SHIPMENT)
RESULT _ $\pi_{\text{Order\#}}$ (LATE_SHIPPED)

e. NEW_YORK _ $\pi_{\text{Warehouse\#}}$ (_ $_{\text{City = 'New York'}}$ (WAREHOUSE))
RESULT _ $\pi_{\text{Order\#, Warehouse\#}}$ (SHIPMENT) ÷ NEW_YORK

5.14 Consider the following relations for a database that keeps track of business trips of salespersons in a sales office:

SALESPERSON (SSN, Name, Start_Year, Dept_No)
TRIP (SSN, From_City, To_City, Departure_Date, Return_Date, Trip_ID)
EXPENSE (Trip_ID, Account#, Amount)

Specify the foreign keys for this schema, stating any assumptions you make.

Answer:

The schema of this question has the following two foreign keys:
1. the attribute SSN of relation TRIP that references relation SALESPERSON, and
2. the attribute Trip_ID of relation EXPENSE that references relation TRIP.
In addition, the attributes Dept_No of relation SALESPERSON and Account# of relation EXPENSE are probably also foreign keys referencing other relations of the database not

mentioned in the question. We now give the queries in relational algebra:

a. $\text{COSTLY\_TRIPS} \_ \__{\text{SUM\_Amount} > 2000} ({}_{\text{Trip\_Id}} \text{F}_{\text{SUM Amount}} (\text{EXPENSE}))$
$\text{RESULT} \_ \text{TRIP} * \pi_{\text{Trip\_Id}} (\text{COSTLY\_TRIPS})$

b. $\text{RESULT} \_ \pi_{\text{SSN}} (\__{\text{To\_City} = \text{'Honolulu'}} (\text{TRIP}))$

c. $\text{RESULT} \_ \text{F}_{\text{SUM Amount}} (\__{\text{SSN} = \text{'234-56-7890'}} (\text{TRIP}) * \text{EXPENSE})$

5.15 Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

STUDENT (<u>SSN</u>, Name, Major, Bdate)
COURSE (<u>Course#</u>, <u>Quarter</u>, Grade)
ENROLL (SSN, <u>Course#</u>, <u>Quarter</u>, Grade)
BOOK_ADOPTION (<u>Course#</u>, <u>Quarter</u>, Book_ISBN)
TEXT (<u>Book_ISBN</u>, Book_Title, Publisher, Author)

Specify the foreign keys for this schema, stating any assumptions you make.

Answer:

The schema of this question has the following four foreign keys:
3. the attribute SSN of relation ENROLL that references relation STUDENT,
4. the attribute Course# in relation ENROLL that references relation COURSE,
5. the attribute Course# in relation BOOK_ADOPTION that references relation COURSE, and
6. the attribute Book_ISBN of relation BOOK_ADOPTION that references relation TEXT.
We now give the queries in relational algebra:

d. $\text{COURSES\_JS\_W99} \_ \; \__{\text{Name = 'John Smith'}} (\text{STUDENT}) * \__{\text{Quarter = 'W99'}} (\text{ENROLL})$
$\text{RESULT} \_ \; \mathcal{F}_{\text{COUNT Course\#}} (\text{COURSES\_JS\_W99})$

e. $\text{CS\_ADOPTIONS} \_ \; \pi_{\text{Course\#, Book\_ISBN}} (\__{\text{Dept = 'CS'}} (\text{COURSE}) * \text{BOOK\_ADOPTION})$
$\text{BOOK\_COUNT} \_ \; {}_{\text{Course\#}} \mathcal{F}_{\text{COUNT Book\_ISBN}} (\text{CS\_ADOPTIONS})$
$\text{COURSES\_NEEDED} \_ \; \pi_{\text{Course\#}} (\__{\text{COUNT\_Book\_ISBN} > 2} (\text{BOOK\_COUNT}))$
$\text{RESULT} \_ \; \pi_{\text{Course\#, Book\_ISBN, Book\_Title}} (\text{COURSES\_NEEDED} * \text{BOOK\_ADOPTION} * \text{TEXT})$

f. $\text{DEPT\_PUBS} \_ \; \pi_{\text{Dept, Publisher}} (\text{COURSE} * \text{BOOK\_ADOPTION} * \text{TEXT})$
$\text{RESULT} \_ \; \pi_{\text{Dept}} (\text{COURSE}) \_ \; \pi_{\text{Dept}} (\__{\text{Publisher} \neq \text{'BC Publishing'}} (\text{DEPT\_PUBS}))$

It is interesting to observe that, contrary to intuition, the above expression does not employ the DIVISION operation, despite the fact that the query involves the word *all*. If, however, the query were to list any department that has adopted *all* books

published by 'BC Publishing', then the solution would be the following expression involving DIVISION:

$\text{DEPT\_BOOKS} \_ \; \pi_{\text{Dept, Book\_ISBN}} (\text{COURSE} * \text{BOOK\_ADOPTION})$
$\text{BOOKS\_BY\_BC} \_ \; \pi_{\text{Book\_ISBN}} (\__{\text{Publisher = 'BC Publishing'}} (\text{TEXT}))$
$\text{RESULT} \_ \; \text{DEPT\_BOOKS} \quad \text{BOOKS\_BY\_BC}$

CHAPTER 6: THE RELATIONAL ALGEBRA AND RELATIONAL CALCULUS

Answers to Selected Exercises

6.15 Show the result of each of the example queries in Section 6.5 if they are applied to the database of Figure 5.6.

Answer:

(QUERY 1) Find the name and address of all employees who work for the 'Research' department.
Result: 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.
Result:
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.
Result: (empty because no tuples satisfy the result).
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.
Result: PNO
1
2

(QUERY 5) List the names of all employees with two or more dependents.
Result: LNAME FNAME
Smith John
Wong Franklin

(QUERY 6) List the names of employees who have no dependents.
Result: 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.

Result: LNAME FNAME
Wallace Jennifer
Wong Franklin

6.16 Specify the following queries on the database schema shown in Figure 5.5 using the relational operators discussed in this chapter. Also show the result of each query if applied to the database of Figure 5.6.

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

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

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

(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 employees who work on every project.

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

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

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

(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.

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

Answers:

In the relational algebra, as in other languages, it is possible to specify the same query in multiple ways. We give one possible solution for each query. We use the symbol s for SELECT, P for PROJECT, J for EQUIJOIN, * for NATURAL JOIN, and f for FUNCTION.

(a) EMP_W_X <-- (s PNAME='ProductX' (PROJECT)) J (PNUMBER),(PNO) (WORKS_ON)
EMP_WORK_10 <-- (EMPLOYEE) J (SSN),(ESSN) (s HOURS>10 (EMP_W_X))
RESULT <-- P LNAME,FNAME (s DNO=5 (EMP_WORK_10))

Result:
LNAME FNAME
Smith John
English Joyce

(b) E <-- (EMPLOYEE) J (SSN,FNAME),(ESSN,DEPENDENT_NAME)
(DEPENDENT)
R <-- P LNAME,FNAME (E)

Result (empty):
LNAME FNAME

(c) WONG_SSN <-- P SSN (s FNAME='Franklin' AND
LNAME='Wong' (EMPLOYEE))
WONG_EMPS <-- (EMPLOYEE) J (SUPERSSN),(SSN) (WONG_SSN)
RESULT <-- P LNAME,FNAME (WONG_EMPS)

Result:

LNAME	FNAME
Smith	John
Narayan	Ramesh
English	Joyce

(d) PROJ_HOURS(PNO,TOT_HRS) <-- PNO f SUM HOURS (WORKS_ON)
RESULT <-- P PNAME,TOT_HRS ((PROJ_HOURS) J (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) PROJ_EMPS(PNO,SSN) <-- P PNO,ESSN (WORKS_ON)
ALL_PROJS(PNO) <-- P PNUMBER (PROJECT)
EMPS_ALL_PROJS <-- PROJ_EMPS -:- ALLPROJS (* DIVISION operation *)
RESULT <-- P LNAME,FNAME (EMPLOYEE * EMP_ALL_PROJS)

Result (empty):
LNAME FNAME

(f) ALL_EMPS <-- P SSN (EMPLOYEE)
WORKING_EMPS(SSN) <-- P ESSN (WORKS_ON)
NON_WORKING_EMPS <-- ALL_EMPS - WORKING_EMPS (* DIFFERENCE
*)
RESULT <-- P LNAME,FNAME (EMPLOYEE * NON_WORKING_EMPS)
Result (empty):
LNAME FNAME

(g) DEPT_AVG_SALS(DNUMBER,AVG_SAL) <-- DNO f AVG SALARY

(EMPLOYEE)
RESULT <-- P DNUMBER,AVG_SAL (DEPT_AVG_SALS * DEPARTMENT)
Result:
DNUMBER AVG_SAL
Research 33250
Administration 31000
Headquarters 55000

(h) RESULT(AVG_F_SAL) <-- f AVG SALARY (s SEX='F' (EMPLOYEE))

Result:
AVG_F_SAL
31000

(i) E_P_HOU(SSN) <--
P ESSN (WORKS_ON J(PNO),(PNUMBER) (s PLOCATION='Houston' (PROJECT)))
D_NO_HOU <--
P DNUMBER (DEPARTMENT) - P DNUMBER (s DLOCATION='Houston' (DEPARTMENT))
E_D_NO_HOU <-- P SSN (EMPLOYEE J(PNO),(DNUMBER) (D_NO_HOU))
RESULT_EMPS <-- E_P_HOU - E_D_NO_HOU (* this is set DIFFERENCE *)
RESULT <-- P LNAME,FNAME,ADDRESS (EMPLOYEE * RESULT_EMPS)

Result:
LNAME FNAME ADDRESS
Wallace Jennifer 291 Berry, Bellaire, TX

(j) DEPT_MANAGERS(SSN)<-- P MGRSSN (DEPARTMENT)
EMPS_WITH_DEPENDENTS(SSN) <-- P ESSN (DEPENDENT)
RESULT_EMPS <-- DEPT_MANAGERS - EMPS_WITH_DEPENDENTS
RESULT <-- P LNAME,FNAME (EMPLOYEE * RESULT_EMPS)

Result:
LNAME FNAME
Borg James

6.18 Consider the LIBRARY relational schema shown in Figure 7.20, which is used to keep track of books, borrowers, and book loans. Referential integrity constraints are shown as directed arcs in Figure 7.20, as in the notation of Figure 7.7. Write down relational expressions for the following queries on the LIBRARY database:

(a) How many copies of the book titled The Lost Tribe are owned by the library branch whose name is "Sharpstown"?

(b) How many copies of the book titled The Lost Tribe are owned by each library branch?
(c) Retrieve the names of all borrowers who do not have any books checked out.

(d) For each book that is loaned out from the "Sharpstown" branch and whose DueDate is today, retrieve the book title, the borrower's name, and the borrower's address.

(e) For each library branch, retrieve the branch name and the total number of books loaned out from that branch.

(f) Retrieve the names, addresses, and number of books checked out for all borrowers who have more than five books checked out.

(g) For each book authored (or co-authored) by "Stephen King", retrieve the title and the number of copies owned by the library branch whose name is "Central".

Answer: (Note: We will use S for SELECT, P for PROJECT, * for NATURAL JOIN, - for SET DIFFERENCE, F for AGGREGATE FUNCTION)

(a) A <-- BOOKCOPIES * LIBRARY-BRANCH * BOOK
RESULT <-- P No_Of_Copies (S BranchName='Sharpstown' and Title='The Lost Tribe'
(A))
Note: A better query would be to do the SELECTs before the JOIN as follows:
A <-- P No_Of_Copies ((S BranchName='Sharpstown' (LIBRARY-BRANCH)) *
(BOOKCOPIES * (S Title='The Lost Tribe'
(BOOK))))

(b) P BranchID,No_Of_Copies ((S Title='The Lost Tribe' (BOOK)) * BOOKCOPIES)

(c) NO_CHECKOUT_B <-- P CardNo (BORROWER) - P CardNo (BOOK_LOANS)
RESULT <-- P Name (BORROWER * NO_CHECKOUT_B)

(d) S <-- P BranchId (S BranchName='Sharpstown' (LIBRARY-BRANCH))
B_FROM_S <-- P BookId,CardNo ((S DueDate='today' (BOOKLOANS)) * S)
RESULT <-- P Title,Name,Address (BOOK * BORROWER * B_FROM_S)

(e) R(BranchId,Total) <-- BranchId FCOUNT(BookId,CardNo) (BOOK_LOANS)
RESULT <-- P BranchName,Total (R * LIBRARY_BRANCH)

(f) B(CardNo,TotalCheckout) <-- CardNo F COUNT(BookId) (BOOK_LOANS)
B5 <-- S TotalCheckout > 5 (B)
RESULT <-- P Name,Address,TotalCheckout (B5 * BORROWER)

(g) SK(BookId,Title) <-- (sAuthorName='Stephen King' (BOOK_AUTHORS)) * BOOK
CENTRAL(BranchId) <-- sBranchName='Central' (LIBRARY_BRANCH)
RESULT <-- P Title,NoOfCopies (SK * BOOKCOPIES * CENTRAL)

6.22
(a)

P	Q	R	A	B	C
10	a	5	10	b	6
10	a	5	10	b	5
25	a	6	25	c	3

(b)

P	Q	R	A	B	C
15	b	8	10	b	6
15	b	8	10	b	5

(c)

P	Q	R	A	B	C
10	a	5	10	b	6
10	a	5	10	b	5
15	b	8	null	null	null
25	a	6	25	c	3

(d)

P	Q	R	A	B	C
15	b	8	10	b	6
null	null	null	25	c	3
15	b	8	10	b	5

(e)

P	Q	R
10	a	5
15	b	8
25	a	6
10	b	6
25	c	3
10	b	5

(f)

P	Q	R	A	B	C
10	a	5	10	b	5

6.24 Specify queries (a), (b), (c), (e), (f), (i), and (j) of Exercise 6.16 in both the
tuple relational calculus and the domain relational calculus.

Answer:

(a) Retrieve the names of employees in department 5 who work more than 10 hours per
week on the 'ProductX' project.
Tuple relational Calculus:
{ e.LNAME, e.FNAME | EMPLOYEE(e) AND e.DNO=5 AND (EXISTS p)
(EXISTS w)
(
WORKS_ON(w) AND PROJECT(p) AND e.SSN=w.ESSN AND
w.PNO=p.PNUMBER AND
p.PNAME='ProductX' AND w.HOURS>10) }
Domain relational Calculus:
{ qs | EMPLOYEE(qrstuvwxyz) AND z=5 AND (EXISTS a) (EXISTS b) (EXISTS
e)
(EXISTS f)
(EXISTS g) (WORKS_ON(efg) AND PROJECT(abcd) AND t=e AND f=b AND
a='ProductX' AND
g>10) }

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

{ e.LNAME, e.FNAME | EMPLOYEE(e) AND (EXISTS d) (DEPENDENT(d) AND e.SSN=d.ESSN
AND e.FNAME=d.DEPENDENT_NAME) }
Domain relational Calculus:
{ qs | (EXISTS t) (EXISTS a) (EXISTS b) (EMPLOYEE(qrstuvwxyz) AND DEPENDENT(abcde)
AND a=t AND b=q) }

(c) Find the names of employees that are directly supervised by 'Franklin Wong'.
Tuple relational Calculus:
{ e.LNAME, e.FNAME | EMPLOYEE(e) AND (EXISTS s) (EMPLOYEE(s) AND s.FNAME='Franklin' AND s.LNAME='Wong' AND e.SUPERSSN=s.SSN) }
Domain relational Calculus:
{ qs | (EXISTS y) (EXISTS a) (EXISTS c) (EXISTS d) (EMPLOYEE(qrstuvwxyz) AND
EMPLOYEE(abcdefghij) AND a='Franklin' AND c='Wong' AND y=d) }

(e) Retrieve the names of employees who work on every project.
Tuple relational Calculus:
{ e.LNAME, e.FNAME | EMPLOYEE(e) AND (FORALL p) (NOT(PROJECT(p)) OR
(EXISTS w) (
WORKS_ON(w) AND p.PNUMBER=w.PNO AND w.ESSN=e.SSN)) }
Domain relational Calculus:
{ qs | (EXISTS t) (EMPLOYEE(qrstuvwxyz) AND (FORALL b) (NOT(PROJECT(abcd)) OR
(EXISTS e) (EXISTS f) (WORKS_ON(efg) AND e=t AND f=b)) }

(f) Retrieve the names of employees who do not work on any project.
Tuple relational Calculus:
{ e.LNAME, e.FNAME | EMPLOYEE(e) AND NOT(EXISTS w) (WORKS_ON(w) AND
w.ESSN=e.SSN) }
Domain relational Calculus:
{ qs | (EXISTS t) (EMPLOYEE(qrstuvwxyz) AND NOT(EXISTS a) (WORKS_ON(abc) AND a=t)
) }

(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.
Tuple relational Calculus:
{ e.LNAME, e.FNAME, e.ADDRESS | EMPLOYEE(e) AND (EXISTS p) (EXISTS w) (
WORKS_ON(w) AND PROJECT(p) AND e.SSN=w.ESSN AND w.PNO=p.PNUMBER AND
p.PLOCATION='Houston' AND NOT(EXISTS l) (DEPT_LOCATIONS(l) AND e.DNO=l.DNUMBER
AND l.DLOCATION='Houston')) }
Domain relational Calculus:

{ qsv | (EXISTS t) (EXISTS z) (EMPLOYEE(qrstuvwxyz) AND (EXISTS b) (EXISTS c)
(EXISTS e)
(EXISTS f) (WORKS_ON(efg) AND PROJECT(abcd) AND t=e AND f=b AND c='Houston' AND
NOT(EXISTS h) NOT(EXISTS i) (DEPT_LOCATIONS(hi) AND z=h AND i='Houston'
)) }

(j) List the last names of department managers who have no dependents.
Tuple relational Calculus:
{ e.LNAME | EMPLOYEE(e) AND (EXISTS d) (DEPARTMENT(d) AND e.SSN=d.MGRSSN AND
NOT(EXISTS x) (DEPENDENT(x) AND e.SSN=x.ESSN)) }
Domain relational Calculus:
{ s | (EXISTS t) (EMPLOYEE(qrstuvwxyz) AND (EXISTS c) (DEPARTMENT(abcd)
AND t=c
AND NOT(EXISTS e) (DEPENDENT(efghi) AND e=t)) }

6.25 No solution provided.

6.26 Specify queries of Exercise 6.18 in both the tuple relational calculus and the domain relational calculus. Also specify these queries in the relational algebra.

Answer:

We will only do tuple and domain relational calculus here.

(a) Retrieve the names of all senior students majoring in 'COSC' (computer science).
Tuple relational Calculus:
{ s.Name | STUDENT(s) AND AND s.Class=5 AND s.Major='COSC' }
Domain relational Calculus:
{ a | (EXISTS c) (EXISTS d) (STUDENT(abcd) AND c=5 AND d='COSC') }

(b) Retrieve the names of all courses taught by professor King in 85 and 86.
Tuple relational Calculus:
{ c.CourseName | COURSE(c) AND (EXISTS s) (SECTION(s) AND c.CourseNumber=s.CourseNumber AND s.Instructor='King' AND (s.Year='85' OR
s.Year='86')) }
Domain relational Calculus:
{ a | (EXISTS b) (EXISTS f) (EXISTS h) (EXISTS i) (COURSE(abcd) AND SECTION(efghi) AND
f=b AND i='King' AND (h='85' OR h='86')) }

(c) For each section taught by professor King, retrieve the course number, semester, year, and number of students who took the section.
This query cannot be done in basic relational calculus as it requires a COUNT function.

(d) Retrieve the name and transcript of each senior student (Class=5) majoring in
COSC. Transcript includes course name, course number, credit hours, semester, year,
and grade for each course completed by the student.
Tuple relational Calculus:
{s.Name, c.CourseName, c.CourseNumber, c.CreditHours, t.Semester, t.Year,
g.Grade |
STUDENT(s)
AND COURSE(c) AND SECTION(t) AND GRADE_REPORT(g) AND s.Class=5
AND
s.Major='COSC' AND s.StudentNumber=g.StudentNumber AND
g.SectionIdentifier=t.SectionIdentifier
AND t.CourseNumber=c.CourseNumber}
Domain relational Calculus:
{aefgklp | (EXISTS b) (EXISTS c) (EXISTS d) (EXISTS n) (EXISTS o) (EXISTS j)
(EXISTS i)
(STUDENT(abcd) AND COURSE(efgh) AND SECTION(ijklm) AND
GRADE_REPORT(nop) AND
c=5 AND d='COSC' AND b=n AND i=o AND j=f)}

(e) Retrieve the names and major departments of all straight A students (students who
have a grade of A in all their courses).
Tuple relational Calculus:
{ s.Name, s.Major | STUDENT(s) AND (FORALL g) (NOT(GRADE_REPORT(g))
OR
NOT(s.StudentNumber=g.StudentNumber) OR g.Grade='A') }
Domain relational Calculus:
{ ad | (EXISTS b) (STUDENT(abcd) AND (FORALL e) (FORALL g) (
NOT(GRADE_REPORT(efg)) OR NOT(b=e) OR g='A')) }

(f) Retrieve the names and major departments of all students who do not have any grade
of A in any of their courses.
Tuple relational Calculus:
{ s.Name, s.Major | STUDENT(s) AND NOT(EXISTS g) (GRADE_REPORT(g)
AND
s.StudentNumber=g.StudentNumber AND g.Grade='A') }
Domain relational Calculus:
{ ad | (EXISTS b) (STUDENT(abcd) AND NOT(EXISTS e) NOT(EXISTS g) (
GRADE_REPORT(efg)
AND b=e AND g='A')) }

6.27 In a tuple relational calculus query with n tuple variables, what would be the
typical minimum number of join conditions? Why? What is the effect of having a
smaller number of join conditions?

Answer:

Typically, there should be at least (n-1) join conditions; otherwise, a cartesian product
with one of the range relations would be taken, which usually does not make sense.

6.28 Rewrite the domain relational calculus queries that followed Q0 in Section 6.7 in

the style of the abbreviated notation of Q0A, where the objective is to minimize
the number of domain variables by writing constants in place of variables
wherever possible.

Answer:

Q1A: { qsv | (EXISTS z) (EXISTS m) (EMPLOYEE(q,r,s,t,u,v,w,x,y,z) AND
DEPARTMENT('Research',m,n,o) AND m=z) }
Q2A: { iksuv | (EXISTS m) (EXISTS n) (EXISTS t) (PROJECT(h,i,'Stafford',k)
AND
EMPLOYEE(q,r,s,t,u,v,w,x,y,z) AND DEPARTMENT(l,m,n,o)) }
The other queries will not be different since they have no constants (no selection
conditions; only join conditions)

6.30 Show how you may specify the following relational algebra operations in both
tuple and domain relational calculus.

(a) SELECT A=c (R(A, B, C)):

(b) PROJECT <A, B> (R(A, B, C)):

(c) R(A, B, C) NATURAL JOIN S(C, D, E):

(d) R(A, B, C) UNION S(A, B, C):

(e) R(A, B, C) INTERSECT S(A, B, C):

(f) R(A, B, C) MINUS S(A, B, C):

(g) R(A, B, C) CARTESIAN PRODUCT S(D, E, F):

(h) R(A, B) DIVIDE S(A):

Answer:

For each operation, we give the tuple calculus expression followed by the domain
calculus expression.

(a) { t | R(t) AND t.A=c}, { xyz | R(xyz) AND x=c }

(b) { t.A, t.B | R(t) }, { xy | R(xyz) }

(c) {t.A, t.B, t.C, q.D, q.E | R(t) AND S(q) AND t.C=q.C },
{ xyzvw | R(xyz) AND (EXISTS u) (S(uvw) AND z=u) }

(d) { t | R(t) OR S(t) },
{ xyz | R(xyz) OR S(xyz) }

(e) { t | R(t) AND S(t) },
{ xyz | R(xyz) AND S(xyz) }

(f) { t | R(t) AND NOT(S(t)) },
{ xyz | R(xyz) AND NOT(S(xyz)) }

(g) { t.A, t.B, t.C, q.D, q.E, q.F | R(t) AND S(q) },
(xyzuvw | R(xyz) AND S(uvw) }

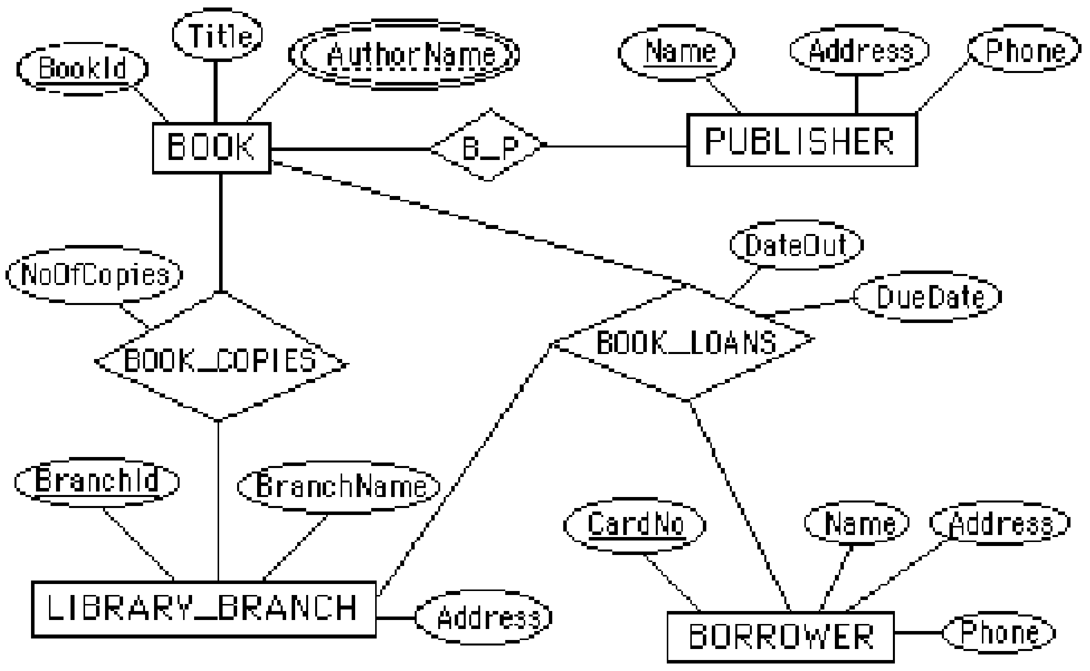
(h) { t.B | R(t) AND (FORALL s)
(NOT(S(s)) OR (EXISTS q) (R(q) AND s.A=q.A AND q.B=t.B)) },
{ y | R(xy) AND (FORALL z) (NOT(S(z)) OR (EXISTS u) (R(uy) AND z=u) }

CHAPTER 7: RELATIONAL DATABASE DESIGN BY ER- AND EER-TO-RELATIONAL MAPPING

Answers to Selected Exercises

7.3 Try to map the relational schema of Figure 6.12 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.

Answer:



Note: We represented BOOK_AUTHORS as a multi-valued attribute of BOOK in the above ER diagram. Alternatively, it can be represented as a weak entity type.

7.4 Figure 7.7 shows an ER schema for a database that may be used to keep track of transport ships and their locations for maritime authorities. Map this schema into a relational schema, and specify all primary keys and foreign keys.

Answer:

SHIP
SNAME OWNER TYPE PNAME
SHIP_TYPE
TYPE TONNAGE HULL
STATE_COUNTRY
NAME CONTINENT
SEAOCEANLAKE
NAME

SHIP_MOVEMENT
SSNAME DATE TIME LONGITUDE LATITUTE
PORT
S_C_NAME PNAME S_O_L_NAME
VISIT
VSNAME VPNAME STARTDATE ENDDATE
f.k.
f.k.
f.k. f.k. f.k.
f.k.
f.k.

7.5 Map the BANK ER schema of Exercise 3.23 (shown in Figure 3.17) into a
relational schema. Specify all primary keys and foreign keys. Repeat for the AIRLINE
schema (Figure 3.16) of Exercise 3.19 and for the other schemas for Exercises 3.16 through
3.24.

Partial Answer:

BANK
CODE NAME ADDR
ACCOUNT
ACCTNO BALANCE TYPE BCODE BNO
CUSTOMER
SSN NAME PHONE ADDR
LOAN
LOANNO AMOUNT TYPE BCODE BNO
BANK_BRANCH
BCODE BRANCHNO ADDR
A_C
SSN ACCTNO
L_C
SSN LOANNO
f.k.
f.k. f.k.
f.k.
f.k. f.k.
f.k.
f.k. f.k.

CHAPTER 8: SQL-99: SCHEMA DEFINITION, BASIC CONSTRAINTS, AND QUERIES

Answers to Selected Exercises

8. 7 Consider the database shown in Figure 1.2, whose schema is shown in Figure 2.1.
What are the referential integrity constraints that should hold on the schema?
Write appropriate SQL DDL statements to define the database.

Answer:

The following referential integrity constraints should hold (we use the notation:
R.(A1, ..., An) --> S.(B1, ..., Bn)
to represent a foreign key from the attributes A1, ..., An of R (the referencing relation)
to S (the referenced relation)):
PREREQUISITE.(CourseNumber) --> COURSE.(CourseNumber)
PREREQUISITE.(PrerequisiteNumber) --> COURSE.(CourseNumber)
SECTION.(CourseNumber) --> COURSE.(CourseNumber)
GRADE_REPORT.(StudentNumber) --> STUDENT.(StudentNumber)
GRADE_REPORT.(SectionIdentifier) --> SECTION.(SectionIdentifier)
One possible set of CREATE TABLE statements to define the database is given below.

```
CREATE TABLE STUDENT ( Name VARCHAR(30) NOT NULL,
StudentNumber INTEGER NOT NULL,
Class CHAR NOT NULL,
Major CHAR(4),
PRIMARY KEY (StudentNumber) );
CREATE TABLE COURSE ( CourseName VARCHAR(30) NOT NULL,
CourseNumber CHAR(8) NOT NULL,
CreditHours INTEGER,
Department CHAR(4),
PRIMARY KEY (CourseNumber),
UNIQUE (CourseName) );
CREATE TABLE PREREQUISITE ( CourseNumber CHAR(8) NOT NULL,
PrerequisiteNumber CHAR(8) NOT NULL,
PRIMARY KEY (CourseNumber, PrerequisiteNumber),
FOREIGN KEY (CourseNumber) REFERENCES
COURSE (CourseNumber),
FOREIGN KEY (PrerequisiteNumber) REFERENCES
COURSE (CourseNumber) );
CREATE TABLE SECTION ( SectionIdentifier INTEGER NOT NULL,
CourseNumber CHAR(8) NOT NULL,
Semester VARCHAR(6) NOT NULL,
Year CHAR(4) NOT NULL,
Instructor VARCHAR(15),
PRIMARY KEY (SectionIdentifier),
FOREIGN KEY (CourseNumber) REFERENCES
COURSE (CourseNumber) );
CREATE TABLE GRADE_REPORT ( StudentNumber INTEGER NOT NULL,
SectionIdentifier INTEGER NOT NULL,
Grade CHAR,
PRIMARY KEY (StudentNumber, SectionIdentifier),
FOREIGN KEY (StudentNumber) REFERENCES
STUDENT (StudentNumber),
FOREIGN KEY (SectionIdentifier) REFERENCES
SECTION (SectionIdentifier) );
```

8. 8 Repeat Exercise 8.7, but use the AIRLINE schema of Figure 5.8.
Answer:

The following referential integrity constraints should hold:
FLIGHT_LEG.(FLIGHT_NUMBER) --> FLIGHT.(NUMBER)
FLIGHT_LEG.(DEPARTURE_AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
FLIGHT_LEG.(ARRIVAL_AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
LEG_INSTANCE.(FLIGHT_NUMBER, LEG_NUMBER) -->
FLIGHT_LEG.(FLIGHT_NUMBER, LEG_NUMBER)
LEG_INSTANCE.(AIRPLANE_ID) --> AIRPLANE.(AIRPLANE_ID)
LEG_INSTANCE.(DEPARTURE_AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
LEG_INSTANCE.(ARRIVAL_AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
FARES.(FLIGHT_NUMBER) --> FLIGHT.(NUMBER)
CAN_LAND.(AIRPLANE_TYPE_NAME) --> AIRPLANE_TYPE.(TYPE_NAME)
CAN_LAND.(AIRPORT_CODE) --> AIRPORT.(AIRPORT_CODE)
AIRPLANE.(AIRPLANE_TYPE) --> AIRPLANE_TYPE.(TYPE_NAME)
SEAT_RESERVATION.(FLIGHT_NUMBER, LEG_NUMBER, DATE) -->
LEG_INSTANCE.(FLIGHT_NUMBER, LEG_NUMBER, DATE)
One possible set of CREATE TABLE statements to define the database is given below.

```
CREATE TABLE AIRPORT ( AIRPORT_CODE CHAR(3) NOT NULL,
NAME VARCHAR(30) NOT NULL,
CITY VARCHAR(30) NOT NULL,
STATE VARCHAR(30),
PRIMARY KEY (AIRPORT_CODE) );
CREATE TABLE FLIGHT ( NUMBER VARCHAR(6) NOT NULL,
AIRLINE VARCHAR(20) NOT NULL,
WEEKDAYS VARCHAR(10) NOT NULL,
PRIMARY KEY (NUMBER) );
CREATE TABLE FLIGHT_LEG ( FLIGHT_NUMBER VARCHAR(6) NOT NULL,
LEG_NUMBER INTEGER NOT NULL,
DEPARTURE_AIRPORT_CODE CHAR(3) NOT NULL,
SCHEDULED_DEPARTURE_TIME TIMESTAMP WITH TIME ZONE,
ARRIVAL_AIRPORT_CODE CHAR(3) NOT NULL,
SCHEDULED_ARRIVAL_TIME TIMESTAMP WITH TIME ZONE,
PRIMARY KEY (FLIGHT_NUMBER, LEG_NUMBER),
FOREIGN KEY (FLIGHT_NUMBER) REFERENCES FLIGHT (NUMBER),
FOREIGN KEY (DEPARTURE_AIRPORT_CODE) REFERENCES
AIRPORT (AIRPORT_CODE),
FOREIGN KEY (ARRIVAL_AIRPORT_CODE) REFERENCES
AIRPORT (AIRPORT_CODE) );
CREATE TABLE LEG_INSTANCE ( FLIGHT_NUMBER VARCHAR(6) NOT NULL,
LEG_NUMBER INTEGER NOT NULL,
LEG_DATE DATE NOT NULL,
NO_OF_AVAILABLE_SEATS INTEGER,
AIRPLANE_ID INTEGER,
DEPARTURE_AIRPORT_CODE CHAR(3),
DEPARTURE_TIME TIMESTAMP WITH TIME ZONE,
ARRIVAL_AIRPORT_CODE CHAR(3),
ARRIVAL_TIME TIMESTAMP WITH TIME ZONE,
PRIMARY KEY (FLIGHT_NUMBER, LEG_NUMBER, LEG_DATE),
FOREIGN KEY (FLIGHT_NUMBER, LEG_NUMBER) REFERENCES
FLIGHT_LEG (FLIGHT_NUMBER, LEG_NUMBER),
FOREIGN KEY (AIRPLANE_ID) REFERENCES
AIRPLANE (AIRPLANE_ID),
FOREIGN KEY (DEPARTURE_AIRPORT_CODE) REFERENCES
AIRPORT (AIRPORT_CODE),
FOREIGN KEY (ARRIVAL_AIRPORT_CODE) REFERENCES
AIRPORT (AIRPORT_CODE) );
CREATE TABLE FARES ( FLIGHT_NUMBER VARCHAR(6) NOT NULL,
FARE_CODE VARCHAR(10) NOT NULL,
AMOUNT DECIMAL(8,2) NOT NULL,
RESTRICTIONS VARCHAR(200),
PRIMARY KEY (FLIGHT_NUMBER, FARE_CODE),
FOREIGN KEY (FLIGHT_NUMBER) REFERENCES FLIGHT (NUMBER) );
CREATE TABLE AIRPLANE_TYPE ( TYPE_NAME VARCHAR(20) NOT NULL,
MAX_SEATS INTEGER NOT NULL,
COMPANY VARCHAR(15) NOT NULL,
PRIMARY KEY (TYPE_NAME) );
CREATE TABLE CAN_LAND ( AIRPLANE_TYPE_NAME VARCHAR(20) NOT NULL,
AIRPORT_CODE CHAR(3) NOT NULL,
PRIMARY KEY (AIRPLANE_TYPE_NAME, AIRPORT_CODE),
FOREIGN KEY (AIRPLANE_TYPE_NAME) REFERENCES
AIRPLANE_TYPE (TYPE_NAME),
FOREIGN KEY (AIRPORT_CODE) REFERENCES
AIRPORT (AIRPORT_CODE) );
CREATE TABLE AIRPLANE ( AIRPLANE_ID INTEGER NOT NULL,
TOTAL_NUMBER_OF_SEATS INTEGER NOT NULL,
AIRPLANE_TYPE VARCHAR(20) NOT NULL,
PRIMARY KEY (AIRPLANE_ID),
FOREIGN KEY (AIRPLANE_TYPE) REFERENCES AIRPLANE_TYPE (TYPE_NAME) );
CREATE TABLE SEAT_RESERVATION ( FLIGHT_NUMBER VARCHAR(6) NOT NULL,
LEG_NUMBER INTEGER NOT NULL,
LEG_DATE DATE NOT NULL,
SEAT_NUMBER VARCHAR(4),
CUSTOMER_NAME VARCHAR(30) NOT NULL,
CUSTOMER_PHONE CHAR(12),
PRIMARY KEY (FLIGHT_NUMBER, LEG_NUMBER, LEG_DATE, SEAT_NUMBER),
FOREIGN KEY (FLIGHT_NUMBER, LEG_NUMBER, LEG_DATE) REFERENCES
LEG_INSTANCE (FLIGHT_NUMBER, LEG_NUMBER, LEG_DATE) );
```

8.9 Consider the LIBRARY relational database schema of Figure 6.12. Choose the appropriate action (reject, cascade, set to null, set to default) for each referential integrity constraint, both for DELETE of a referenced tuple, and for UPDATE of a primary key attribute value in a referenced tuple. Justify your choices.

Answer:

Below are possible choices. In general, if it is not clear which action to choose, REJECT should be chosen, since it will not permit automatic changes to happen (by update propagation) that may be unintended.
BOOK_AUTHORS.(BookId) --> BOOK.(BookId)
CASCADE on both DELETE or UPDATE (since this corresponds to a multi-valued attribute of BOOK (see the solution to Exercise 6.27); hence, if a BOOK is deleted, or the value of its BookId is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_AUTHORS tuples)
BOOK.(PublisherName) --> PUBLISHER.(Name)
REJECT on DELETE (we should not delete a PUBLISHER tuple which has existing BOOK tuples that reference the PUBLISHER)
CASCADE on UPDATE (if a PUBLISHER's Name is updated, the change should be propagated automatically to all referencing BOOK tuples)
BOOK_LOANS.(BookId) --> BOOK.(BookId)
CASCADE on both DELETE or UPDATE (if a BOOK is deleted, or the value of its BookId is updated (changed), the deletion or change is automatically propagated to the referencing

BOOK_LOANS tuples) (Note: One could also choose REJECT on DELETE)
BOOK_COPIES.(BookId) --> BOOK.(BookId)
CASCADE on both DELETE or UPDATE (if a BOOK is deleted, or the value of its BookId is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_COPIES tuples)
BOOK_LOANS.(CardNo) --> BORROWER.(CardNo)
CASCADE on both DELETE or UPDATE (if a BORROWER tuple is deleted, or the value of its CardNo is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_LOANS tuples) (Note: One could also choose REJECT on DELETE, with the idea that if a BORROWER is deleted, it is necessary first to make a printout of all BOOK_LOANS outstanding before deleting the BORROWER; in this case, the tuples in BOOK_LOANS that reference the BORROWER being deleted would first be explicitly deleted after making the printout, and before the BORROWER is deleted)
BOOK_COPIES.(BranchId) --> LIBRARY_BRANCH.(BranchId)
CASCADE on both DELETE or UPDATE (if a LIBRARY_BRANCH is deleted, or the value of its BranchId is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_COPIES tuples) (Note: One could also choose REJECT on DELETE)
BOOK_LOANS.(BranchId) --> LIBRARY_BRANCH.(BranchId)
CASCADE on both DELETE or UPDATE (if a LIBRARY_BRANCH is deleted, or the value of its BranchId is updated (changed), the deletion or change is automatically propagated to the referencing BOOK_LOANS tuples) (Note: One could also choose REJECT on DELETE)

8.10 Write appropriate SQL DDL statements for declaring the LIBRARY relational database schema of Figure 76.12. Use the referential action chosen in Exercise 8.9.

Answer: One possible set of CREATE TABLE statements is given below:

```
CREATE TABLE BOOK ( BookId CHAR(20) NOT NULL,
Title VARCHAR(30) NOT NULL,
PublisherName VARCHAR(20),
PRIMARY KEY (BookId),
FOREIGN KEY (PublisherName) REFERENCES PUBLISHER (Name) ON UPDATE
CASCADE );
CREATE TABLE BOOK_AUTHORS ( BookId CHAR(20) NOT NULL,
AuthorName VARCHAR(30) NOT NULL,
PRIMARY KEY (BookId, AuthorName),
FOREIGN KEY (BookId) REFERENCES BOOK (BookId)
ON DELETE CASCADE ON UPDATE CASCADE );
CREATE TABLE PUBLISHER ( Name VARCHAR(20) NOT NULL,
Address VARCHAR(40) NOT NULL,
Phone CHAR(12),
PRIMARY KEY (Name) );
CREATE TABLE BOOK_COPIES ( BookId CHAR(20) NOT NULL,
BranchId INTEGER NOT NULL,
No_Of_Copies INTEGER NOT NULL,
PRIMARY KEY (BookId, BranchId),
FOREIGN KEY (BookId) REFERENCES BOOK (BookId)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (BranchId) REFERENCES BRANCH (BranchId)
ON DELETE CASCADE ON UPDATE CASCADE );
CREATE TABLE BORROWER ( CardNo INTEGER NOT NULL,
Name VARCHAR(30) NOT NULL,
Address VARCHAR(40) NOT NULL,
Phone CHAR(12),
PRIMARY KEY (CardNo) );
CREATE TABLE BOOK_LOANS ( CardNo INTEGER NOT NULL,
BookId CHAR(20) NOT NULL,
BranchId INTEGER NOT NULL,
DateOut DATE NOT NULL,
DueDate DATE NOT NULL,
PRIMARY KEY (CardNo, BookId, BranchId),
FOREIGN KEY (CardNo) REFERENCES BORROWER (CardNo)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (BranchId) REFERENCES LIBRARY_BRANCH (BranchId)
ON DELETE CASCADE ON UPDATE CASCADE,
FOREIGN KEY (BookId) REFERENCES BOOK (BookId)
ON DELETE CASCADE ON UPDATE CASCADE );
CREATE TABLE LIBRARY_BRANCH ( BranchId INTEGER NOT NULL,
BranchName VARCHAR(20) NOT NULL,
Address VARCHAR(40) NOT NULL,
PRIMARY KEY (BranchId) );
```

8.11 Write SQL queries for the LIBRARY database queries given in Exercise 6.18.

Answer:

Below, we give one possible SQL query for each request. Other queries are also possible.

(a) How many copies of the book titled The Lost Tribe are owned by the library branch whose name is "Sharpstown"?

```
SELECT NoOfCopies
FROM ( (BOOK NATURAL JOIN BOOK_COPIES ) NATURAL JOIN
LIBRARY_BRANCH )
WHERE Title='The Lost Tribe' AND BranchName='Sharpstown'
```

(b) How many copies of the book titled The Lost Tribe are owned by each library branch?

```
SELECT BranchName, NoOfCopies
FROM ( (BOOK NATURAL JOIN BOOK_COPIES ) NATURAL JOIN
LIBRARY_BRANCH )
WHERE Title='The Lost Tribe'
```

(c) Retrieve the names of all borrowers who do not have any books checked out.

```
SELECT Name
FROM BORROWER B
WHERE NOT EXIST ( SELECT *
FROM BOOK_LOANS L
WHERE B.CardNo = L.CardNo )
```

(d) For each book that is loaned out from the "Sharpstown" branch and whose DueDate is today, retrieve the book title, the borrower's name, and the borrower's address.

```
SELECT B.Title, R.Name, R.Address
FROM BOOK B, BORROWER R, BOOK_LOANS BL, LIBRARY_BRANCH LB
WHERE LB.BranchName='Sharpstown' AND LB.BranchId=BL.BranchId AND
BL.DueDate='today' AND BL.CardNo=R.CardNo AND BL.BookId=B.BookId
```

(e) For each library branch, retrieve the branch name and the total number of books loaned out from that branch.

```
SELECT L.BranchName, COUNT(*)
FROM BOOK_COPIES B, LIBRARY_BRANCH L
WHERE B.BranchId = L.BranchId
GROUP BY L.BranchName
```

(f) Retrieve the names, addresses, and number of books checked out for all borrowers who have more than five books checked out.

```
SELECT B.CardNo, B.Name, B.Address, COUNT(*)
FROM BORROWER B, BOOK_LOANS L
WHERE B.CardNo = L.CardNo
GROUP BY B.CardNo
HAVING COUNT(*) > 5
```

(g) For each book authored (or co-authored) by "Stephen King", retrieve the title and the number of copies owned by the library branch whose name is "Central".

```
SELECT TItle, NoOfCopies
FROM ( ( (BOOK_AUTHORS NATURAL JOIN BOOK) NATURAL JOIN
BOOK_COPIES)
NATURAL JOIN LIBRARY_BRANCH)
WHERE Author_Name = 'Stephen King' and BranchName = 'Central'
```

8.12 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 in an efficient manner when updates are applied to the database?

Answer:

One possible technique that is often used to check efficiently for the key constraint is to create an index on the combination of attributes that form each key (primary or secondary). Before inserting a new record (tuple), each index is searched to check 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.
For checking the foreign key constraint, an index on the primary key of each referenced relation will make this check relatively efficient. Whenever a new record is inserted in a referencing relation , its foreign key value is used to search the index for the primary key of the referenced relation, and if the referenced record exists, then the new record can be successfully inserted in the referencing relation.
For deletion of a referenced record, it is useful to have an index on the foreign key of each referencing relation so as to be able to determine efficiently whether any records reference the record being deleted.
If the indexes described above do not exist, and no alternative access structure (for example, hashing) is used in their place, then it is necessary to do linear searches to check for any of the above constraints, making the checks quite inefficient.

8.13 Specify the queries of Exercise 6.16 in SQL. Show the result of each query if applied to the COMPANY database of Figure 5.6.

Answers:

In SQL, as in most languages, it is possible to specify the same query in
multiple ways. We will give one or more possible specification for each query.

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

```
SELECT LNAME, FNAME
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE DNO=5 AND SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX' AND HOURS>10
```

Another possible SQL query uses nesting as follows:

```
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' ) )
```

Result:

```
LNAME FNAME
Smith John
English Joyce
```

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

```
SELECT LNAME, FNAME
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 )
```

Result (empty):

```
LNAME FNAME
```

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

```
SELECT E.LNAME, E.FNAME
FROM EMPLOYEE E, EMPLOYEE S
WHERE S.FNAME='Franklin' AND S.LNAME='Wong' AND E.SUPERSSN=S.SSN
```

Another possible SQL query uses nesting as follows:

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SUPERSSN IN ( SELECT SSN
FROM EMPLOYEE
WHERE FNAME='Franklin' AND LNAME='Wong' )
```

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.

```
SELECT PNAME, SUM (HOURS)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNAME
```

Result:

```
PNAME SUM(HOURS)
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.

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE NOT EXISTS ( SELECT PNUMBER
FROM PROJECT
WHERE NOT EXISTS ( SELECT *
FROM WORKS_ON
WHERE PNUMBER=PNO AND ESSN=SSN ) )
```

Result (empty):

```
LNAME FNAME
```

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

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE NOT EXISTS ( SELECT *
FROM WORKS_ON
WHERE ESSN=SSN )
```

Result (empty):

```
LNAME FNAME
```

(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
```

Result:

```
DNAME AVG(SALARY)
Research 33250
Administration 31000
Headquarters 55000
```

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

```
SELECT AVG (SALARY)
FROM EMPLOYEE
WHERE SEX='F'
```

Result:

AVG(SALARY)
31000

(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.

```
SELECT LNAME, FNAME, 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' )
```

Result:
LNAME FNAME ADDRESS
Wallace Jennifer 291 Berry, Bellaire, TX

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

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE EXISTS ( SELECT *
FROM DEPARTMENT
WHERE SSN=MGRSSN )
AND
NOT EXISTS ( SELECT *
FROM DEPENDENT
WHERE SSN=ESSN )
```

Result:
LNAME FNAME
Borg James

8.14 Specify the following additional queries on the database of Figure 5.5 in SQL. Show the query results if applied to the database of Figure 5.6.

(a) For each department whose average employee salary is more than $30000, retrieve the department name and the number of employees working for that department.

(b) Suppose we want the number of male employees in each department rather than all employees as in Exercise 8.14(a). Can we specify this query in SQL? Why or why not?

Answers:

(a)
```
SELECT DNAME, COUNT (*)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO
GROUP BY DNAME
HAVING AVG (SALARY) > 30000
```

Result:
DNAME DNUMBER COUNT(*)
Research 5 4
Administration 4 3

Headquarters 1 1

(b) The query may still be specified in SQL by using a nested query as follows (not all implementations may support this type of query):

```
SELECT DNAME, COUNT (*)
FROM DEPARTMENT, EMPLOYEE
WHERE DNUMBER=DNO AND SEX='M' AND DNO IN ( SELECT DNO
FROM EMPLOYEE
GROUP BY DNO
HAVING AVG (SALARY) > 30000 )
GROUP BY DNAME
```

Result:
DNAME DNUMBER COUNT(*)
Research 5 3
Administration 4 1
Headquarters 1 1

8.15 Specify the updates of Exercise 5.10 using the SQL update commands.

Answers:

Below, we show how each of the updates may be specified in SQL. Notice that some of these updates violate integrity constraints as discussed in the solution to Exercise 5.10, and hence should be rejected if executed on the database of Figure 5.6.

(a) Insert < 'Robert', 'F', 'Scott', '943775543', '21-JUN-42', '2365 Newcastle Rd, Bellaire, TX', M, 58000, '888665555', 1 > into EMPLOYEE.

```
INSERT INTO EMPLOYEE
VALUES ('Robert', 'F', 'Scott', '943775543', '21-JUN-42', '2365 Newcastle Rd, Bellaire, TX',
M, 58000, '888665555', 1)
```

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

```
INSERT INTO PROJECT
VALUES ('ProductA', 4, 'Bellaire', 2)
```

(c) Insert < 'Production', 4, '943775543', '01-OCT-88' > into DEPARTMENT.

```
INSERT INTO DEPARTMENT
VALUES ('Production', 4, '943775543', '01-OCT-88')
```

(d) Insert < '677678989', null, '40.0' > into WORKS_ON.

```
INSERT INTO WORKS_ON
VALUES ('677678989', NULL, '40.0')
```

(e) Insert < '453453453', 'John', M, '12-DEC-60', 'SPOUSE' > into DEPENDENT.

```
INSERT INTO DEPENDENT
VALUES ('453453453', 'John', M, '12-DEC-60', 'SPOUSE')
```

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

```
DELETE FROM WORKS_ON
WHERE ESSN= '333445555'
```

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

```
DELETE FROM EMPLOYEE
WHERE SSN= '987654321'
```

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

```
DELETE FROM PROJECT
WHERE PNAME= 'ProductX'
```

(i) Modify the MGRSSN and MGRSTARTDATE of the DEPARTMENT tuple with DNUMBER=
5 to '123456789' and '01-OCT-88', respectively.

```
UPDATE DEPARTMENT
SET MGRSSN = '123456789', MGRSTARTDATE = '01-OCT-88'
WHERE DNUMBER= 5
```

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

```
UPDATE EMPLOYEE
SET SUPERSSN = '943775543'
WHERE SSN= '999887777'
```

(k) Modify the HOURS attribute of the WORKS_ON tuple with ESSN= '999887777' and PNO= 10 to '5.0'.

```
UPDATE WORKS_ON
SET HOURS = '5.0'
WHERE ESSN= '999887777' AND PNO= 10
```

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

(a) Retrieve the names of all senior students majoring in 'COSC' (computer science).

(b) Retrieve the names of all courses taught by professor King in 85 and 86.

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

(d) Retrieve the name and transcript of each senior student (Class=5) majoring in COSC. Transcript includes course name, course number, credit hours, semester, year, and grade for each course completed by the student.

(e) Retrieve the names and major departments of all straight A students (students who have a grade of A in all their courses).

(f) Retrieve the names and major departments of all students who do not have any grade of A in any of their courses.

Answers:

(a)
```
SELECT Name
FROM STUDENT
WHERE Major='COSC'
```

(b)
```
SELECT CourseName
FROM COURSE, SECTION
WHERE COURSE.CourseNumber=SECTION.CourseNumber AND Instructor='King'
AND (Year='85' OR Year='86')
```

Another possible SQL query uses nesting as follows:

```
SELECT CourseName
```

```
FROM COURSE
WHERE CourseNumber IN ( SELECT CourseNumber
FROM SECTION
WHERE Instructor='King' AND (Year='85' OR Year='86') )
```

(c) SELECT CourseNumber, Semester, Year, COUNT(*)
FROM SECTION, GRADE_REPORT
WHERE Instructor='King' AND SECTION.SectionIdentifier=GRADE_REPORT.SectionIdentifier
GROUP BY CourseNumber, Semester, Year

(d) SELECT Name, CourseName, C.CourseNumber, CreditHours, Semester, Year, Grade
FROM STUDENT ST, COURSE C, SECTION S, GRADE_REPORT G
WHERE Class=5 AND Major='COSC' AND ST.StudentNumber=G.StudentNumber AND
G.SectionIdentifier=S.SectionIdentifier AND S.CourseNumber=C.CourseNumber

(e) SELECT Name, Major
FROM STUDENT
WHERE NOT EXISTS (SELECT *
FROM GRADE_REPORT
WHERE StudentNumber= STUDENT.StudentNumber AND NOT(Grade='A'))

(f) SELECT Name, Major
FROM STUDENT
WHERE NOT EXISTS (SELECT *
FROM GRADE_REPORT
WHERE StudentNumber= STUDENT.StudentNumber AND Grade='A')

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

(a) Insert a new student <'Johnson', 25, 1, 'MATH'> in the database.

(b) Change the class of student 'Smith' to 2.

(c) Insert a new course <'Knowledge Engineering','COSC4390', 3,'COSC'>.

(d) Delete the record for the student whose name is 'Smith' and student number is 17.

Answers:

(a) INSERT INTO STUDENT
VALUES ('Johnson', 25, 1, 'MATH')

(b) UPDATE STUDENT
SET CLASS = 2
WHERE Name='Smith'

(c) INSERT INTO COURSE
VALUES ('Knowledge Engineering','COSC4390', 3,'COSC')

(d) DELETE FROM STUDENT
WHERE Name='Smith' AND StudentNumber=17

8.18 no answer provided

CHAPTER 9: MORE SQL: ASSERTIONS, VIEWS, AND PROGRAMMING TECHNIQUES

No exercises.

CHAPTER 10: FUNCTIONAL DEPENDENCIES AND NORMALIZATION FOR RELATIONAL DATABASES

Answers to Selected Exercises

10.17 Suppose we have the following requirements for a university database that is used to keep track of students transcripts:

(a) The university keeps track of each student's name (SNAME), student number (SNUM), social security number (SSSN), current address (SCADDR) and phone (SCPHONE), permanent address (SPADDR) and phone (SPPHONE), birthdate (BDATE), sex (SEX), class (CLASS) (freshman, sophomore, ..., graduate), major department (MAJORDEPTCODE), minor department (MINORDEPTCODE) (if any), and degree program (PROG) (B.A., B.S., ..., Ph.D.). Both ssn and student number have unique values for each student.

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

(c) Each course has a course name (CNAME), description (CDESC), code number (CNUM), number of semester hours (CREDIT), level (LEVEL), and offering department (CDEPT). The value of code number is unique for each course.

(d) Each section has an instructor (INSTUCTORNAME), semester (SEMESTER), year (YEAR), course (SECCOURSE), and section number (SECNUM). Section numbers distinguish different 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.

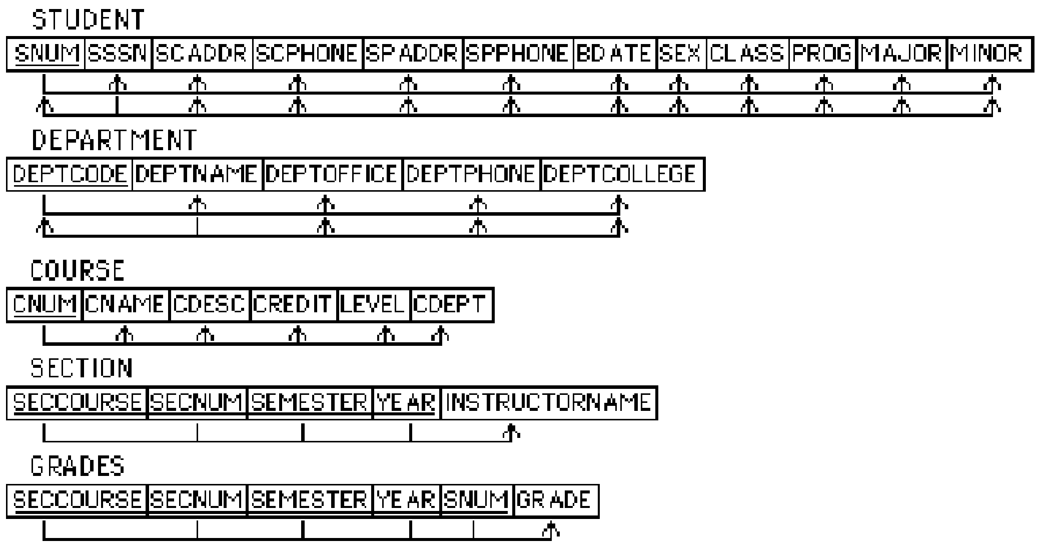
(e) A transcript refers to a student (SSSN), refers to a particular section, and grade (GRADE).

Design an relational database schema for this database application. First show all the functional dependencies that should hold among the attributes. Then, design relation schemas for the database that are each in 3NF or BCNF. Specify the key attributes of each relation. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.

Answer:

From the above description, we can presume that the following functional dependencies hold on the attributes:
FD1: {SSSN} -> {SNAME, SNUM, SCADDR, SCPHONE, SPADDR, SPPHONE, BDATE, SEX, CLASS,
MAJOR, MINOR, PROG}
FD2: {SNUM} -> {SNAME, SSSN, SCADDR, SCPHONE, SPADDR, SPPHONE, BDATE, SEX, CLASS,
MAJOR, MINOR, PROG}
FD3: {DEPTNAME} -> {DEPTCODE, DEPTOFFICE, DEPTPHONE, DEPTCOLLEGE}
FD4: {DEPTCODE} -> {DEPTNAME, DEPTOFFICE, DEPTPHONE, DEPTCOLLEGE}
FD5: {CNUM} -> {CNAME, CDESC, CREDIT, LEVEL, CDEPT}

FD6: {SECCOURSE, SEMESTER, YEAR, SECNUM} -> {INSTRUCTORNAME}
FD7: {SECCOURSE, SEMESTER, YEAR, SECNUM, SSSN} -> {GRADE}
These are the basic FDs that we can define from the given requirements; using inference rules IR1 to IR3, we can deduce many others. FD1 and FD2 refer to student attributes; we can define a relation STUDENT and choose either SSSN or SNUM as its primary key. Similarly, FD3 and FD4 refer to department attributes, with either DEPTNAME or DEPTCODE as primary key. FD5 defines COURSE attributes, and FD6 SECTION attributes. Finally, FD7 defines GRADES attributes. We can create one relation for each of STUDENT, DEPARTMENT, COURSE, SECTION, and GRADES as shown below, where the primary keys are underlined. The COURSE, SECTION, and GRADES relations are in 3NF and BCNF if no other dependencies exist. The STUDENT and DEPARTMENT relations are in 3NF and BCNF according to the general definition given in Sections 14.4 and 14.5, but not according to the definitions of Section 14.3 since both relations have secondary keys.


STUDENT
SNUM SSSN SCADDR SCPHONE SPADDR SPPHONE BDATE SEX CLASS PROG MAJOR MINOR
DEPARTMENT
DEPTCODE DEPTNAME DEPTOFFICE DEPTPHONE DEPTCOLLEGE
COURSE
CNUM CNAME CDESC CREDIT LEVEL CDEPT
SECTION
SECCOURSE SECNUM SEMESTER YEAR INSTRUCTORNAME
GRADES
SECCOURSE SECNUM SEMESTER YEAR SNUM GRADE

The foreign keys will be as follows:
STUDENT.MAJOR -> DEPARTMENT.DEPTCODE
STUDENT.MINOR -> DEPARTMENT.DEPTCODE
COURSE.CDEPT -> DEPARTMENT.DEPTCODE
SECTION.SECCOURSE -> COURSE.CNUM
GRADES.(SECCOURSE, SEMESTER, YEAR, SECNUM) ->
SECTION.(SECCOURSE, SEMESTER, YEAR, SECNUM)
GRADES.SNUM -> STUDENT.SNUM
Note: We arbitrarily chose SNUM over SSSN for primary key of STUDENT, and DEPTCODE over DEPTNAME for primary key of DEPARTMENT.

10.18 Prove or disprove the following inference rules for functional dependencies. A proof can be made either by a proof argument or by using inference rules IR1 through IR3. A disproof should be done by demonstrating a relation instance that satisfies the conditions and functional dependencies in the left hand side of the inference rule but do not satisfy the conditions or dependencies in the right hand side.

(a) {W ->Y, X ->Z} |= {WX ->Y }

(b) {X ->Y} and Z subset-of Y |= { X ->Z }

(c) { X ->Y, X ->W, WY ->Z} |= {X ->Z}

(d) {XY ->Z, Y ->W} |= {XW ->Z}

(e) {X ->Z, Y ->Z} |= {X ->Y}

(f) {X ->Y, XY ->Z} |= {X ->Z}

Answer:

(a) Proof:
(1) W ->Y (given)
(2) X ->Z (given)
(3) WX ->YX (using IR2 (augmentation) to augment (1) with X)
(4) YX ->Y (using IR1 (reflexivity), knowing that Y subset-of YX)
(5) WX ->Y (using IR3 (transitivity) on (3) and (4))

(b) Proof:
(1) X ->Y (given)
(2) Y ->Z (using IR1 (reflexivity), given that Z subset-of Y)
(3) X ->Z (using IR3 (transitivity) on (1) and (2))

(c) Proof:
(1) X ->Y (given)
(2) X ->W (given)
(3) WY ->Z (given)
(4) X ->XY (using IR2 (augmentation) to augment (1) with X)
(5) XY ->WY (using IR2 (augmentation) to augment (2) with Y)
(6) X ->WY (using IR3 (transitivity) on (4) and (5))
(7) X ->Z (using IR3 (transitivity) on (6) and (3))

(d) Disproof: X Y Z W
t 1 =x 1 y 1 z 1 w 1
t 2 =x 1 y 2 z 2 w 1
The above two tuples satisfy XY ->Z and Y ->W but do not satisfy XW ->Z

(e) Disproof: X Y Z
t 1 =x 1 y 1 z 1
t 2 =x 1 y 2 z 1
The above two tuples satisfy X ->Z and Y ->Z but do not satisfy X ->Y

(f) Proof:
(1) X ->Y (given)
(2) XY ->Z (given)
(3) X ->XY (using IR2 (augmentation) to augment (1) with X)
(4) X ->Z (using IR3 (transitivity) on (3) and (2))

10.19 Consider the following two sets of functional dependencies F= {A ->C, AC ->D, E ->AD, E ->H} and G = {A ->CD, E ->AH}. Check whether or not they are equivalent.

Answer:

To show equivalence, we prove that G is covered by F and F is covered by G.
Proof that G is covered by F:
{A} + = {A, C, D} (with respect to F), which covers A ->CD in G
{E} + = {E, A, D, H, C} (with respect to F), which covers E ->AH in G
Proof that F is covered by G:
{A} + = {A, C, D} (with respect to G), which covers A ->C in F
{A, C} + = {A, C, D} (with respect to G), which covers AC ->D in F
{E} + = {E, A, H, C, D} (with respect to G), which covers E ->AD and E ->H in F

10.20 Consider the relation schema EMP_DEPT in Figure 14.3(a) and the following set G of functional dependencies on EMP_DEPT: G = {SSN ->{ENAME, BDATE, ADDRESS, DNUMBER} , DNUMBER ->{DNAME, DMGRSSN} }. Calculate the closures {SSN} + and {DNUMBER} + with respect to G.

Answer:

{SSN} + ={SSN, ENAME, BDATE, ADDRESS, DNUMBER, DNAME, DMGRSSN}
{DNUMBER} + ={DNUMBER, DNAME, DMGRSSN}

10.21 Is the set of functional dependencies G in Exercise 14.20 minimal? If not, try to find an minimal set of functional dependencies that is equivalent to G. Prove that your set is equivalent to G.

Answer:

The set G of functional dependencies in Exercise 14.20 is not minimal, because it violates rule 1 of minimality (every FD has a single attribute for its right hand side).
The set F is an equivalent minimal set: F= {SSN ->{ENAME},SSN ->{BDATE}, SSN->{ADDRESS}, SSN ->{DNUMBER} , DNUMBER ->{DNAME}, DNUMBER->{DMGRSSN}}
To show equivalence, we prove that G is covered by F and F is covered by G.
Proof that G is covered by F:
{SSN} + = {SSN, ENAME, BDATE, ADDRESS, DNUMBER, DNAME, DMGRSSN} (with respect to F), which covers SSN ->{ENAME, BDATE, ADDRESS, DNUMBER} in G
{DNUMBER} + ={DNUMBER, DNAME, DMGRSSN} (with respect to F), which covers DNUMBER ->{DNAME, DMGRSSN} in G
Proof that F is covered by G:
{SSN} + = {SSN, ENAME, BDATE, ADDRESS, DNUMBER, DNAME, DMGRSSN} (with respect to G), which covers SSN ->{ENAME}, SSN ->{BDATE}, SSN ->{ADDRESS}, and SSN ->{DNUMBER} in F
{DNUMBER} + ={DNUMBER, DNAME, DMGRSSN} (with respect to G), which covers DNUMBER ->{DNAME} and DNUMBER->{DMGRSSN} in F

10.22 What update anomalies occur in the EMP_PROJ and EMP_DEPT relations of Figure 14.3 and 14.4?

Answer:

In EMP_PROJ, the partial dependencies {SSN}->{ENAME} and {PNUMBER}->{PNAME, PLOCATION} can cause anomalies. For example, if a PROJECT temporarily has no EMPLOYEEs working on it, its information (PNAME, PNUMBER, PLOCATION) will not be

represented in the database when the last EMPLOYEE working on it is removed (deletion anomaly). A new PROJECT cannot be added unless at least one EMPLOYEE is assigned to work on it (insertion anomaly). Inserting a new tuple relating an existing EMPLOYEE to an existing PROJECT requires checking both partial dependencies; for example, if a different value is entered for PLOCATION than those values in other tuples with the same value for PNUMBER, we get an update anomaly. Similar comments apply to EMPLOYEE information. The reason is that EMP_PROJ represents the relationship between EMPLOYEEs and PROJECTs, and at the same time represents information concerning EMPLOYEE and PROJECT entities.
In EMP_DEPT, the transitive dependency {SSN}->{DNUMBER}->{DNAME, DMGRSSN} can cause anomalies. For example, if a DEPARTMENT temporarily has no EMPLOYEEs working for it, its information (DNAME, DNUMBER, DMGRSSN) will not be represented in the database when the last EMPLOYEE working on it is removed (deletion anomaly). A new DEPARTMENT cannot be added unless at least one EMPLOYEE is assigned to work on it
(insertion anomaly). Inserting a new tuple relating a new EMPLOYEE to an existing DEPARTMENT requires checking the transitive dependencies; for example, if a different value is entered for DMGRSSN than those values in other tuples with the same value for DNUMBER, we get an update anomaly. The reason is that EMP_DEPT represents the relationship between EMPLOYEEs and DEPARTMENTs, and at the same time represents information concerning EMPLOYEE and DEPARTMENT entities.

10.23 In what normal form is the LOTS relation schema in Figure 10.11(a) with the respect to the restrictive interpretations of normal form that take only the primary key into account? Will it be in the same normal form if the general definitions of normal form were used?

Answer:

If we only take the primary key into account, the LOTS relation schema in Figure 14.11 (a) will be in 2NF since there are no partial dependencies on the primary key .
However, it is not in 3NF, since there are the following two transitive dependencies on the primary key:
PROPERTY_ID# ->COUNTY_NAME ->TAX_RATE, and
PROPERTY_ID# ->AREA ->PRICE.
Now, if we take all keys into account and use the general definition of 2NF and 3NF, the LOTS relation schema will only be in 1NF because there is a partial dependency COUNTY_NAME ->TAX_RATE on the secondary key {COUNTY_NAME, LOT#}, which violates 2NF.

10.24 Prove that any relation schema with two attributes is in BCNF.

Answer:

Consider a relation schema R={A, B} with two attributes. The only possible (non-trivial) FDs are {A} ->{B} and {B} ->{A}. There are four possible cases:
(i) No FD holds in R. In this case, the key is {A, B} and the relation satisfies BCNF.
(ii) Only {A} ->{B} holds. In this case, the key is {A} and the relation satisfies BCNF.
(iii) Only {B} ->{A} holds. In this case, the key is {B} and the relation satisfies BCNF.
(iv) Both {A} ->{B} and {B} ->{A} hold. In this case, there are two keys {A} and {B} and the relation satisfies BCNF.
Hence, any relation with two attributes is in BCNF.

10.25 Why do spurious tuples occur in the result of joining the EMP_PROJ1 and EMPLOCS relations of Figure 14.5 (result shown in Figure 14.6)?

Answer:

In EMP_LOCS, a tuple (e, l) signifies that employee with name e works on some project located in location l. In EMP_PROJ1, a tuple (s, p, h, pn, l) signifies that employee with social security number s works on project p that is located at location l. When we join EMP_LOCS with EMP_PROJ1, a tuple (e, l) in EMP_LOCS can be joined with a tuple (s, p, h, pn, l) in EMP_PROJ1 where e is the name of some employee and s is the social security number of a different employee, resulting in spurious tuples. The lossless join property (see Chapter 13) can determine whether or not spurious tuples may result based on the FDs in the two relations being joined.

10.26 Consider the universal relation R = {A, B, C, D, E, F, G, H, I} 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, then 3NF relations.

Answer:

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}

10.27 Repeat exercise 10.26 for the following different set of functional dependencies G = { {A, B} -> {C}, {B, D} -> {E, F}, {A, D} -> {G, H}, {A} -> {I}, {H} -> {J} }.

Answer:

To help in solving this problem systematically, we can first find the closures of all

single attributes to see if any is a key on its own as follows:
{A}+ -> {A, I}, {B}+ -> {B}, {C}+ -> {C}, {D}+ -> {D}, {E}+ -> {E}, {F}+ -> {F},
{G}+ -> {G}, {H}+ -> {H, J}, {I}+ -> {I}, {J}+ -> {J}
Since none of the single attributes is a key, we next calculate the closures of pairs of attributes that are possible keys:
{A, B}+ -> {A, B, C, I}, {B, D}+ -> {B, D, E, F}, {A, D}+ -> {A, D, G, H, I, J}
None of these pairs are keys either since none of the closures includes all attributes. But the union of the three closures includes all the attributes:
{A, B, D}+ -> {A, B, C, D, E, F, G, H, I}
Hence, {A, B, D} is a key. (Note: Algorithm 15.4a (see chapter 15 in the textbook) can be used to determine a key).
Based on the above analysis, we decompose as follows, in a similar manner to problem 14.26, starting with the following relation R:
R = {A, B, D, C, E, F, G, H, I}
The first-level partial dependencies on the key (which violate 2NF) are:
{A, B} -> {C, I}, {B, D} -> {E, F}, {A, D}+ -> {G, H, I, J}
Hence, R is decomposed into R1, R2, R3, R4 (keys are underlined):
R1 = {A, B, C, I}, R2 = {B, D, E, F}, R3 = {A, D, G, H, I, J}, R4 = {A, B, D}
Additional partial dependencies exist in R1 and R3 because {A} -> {I}. Hence, we remove {I} into R5, so the following relations are the result of 2NF decomposition:
R1 = {A, B, C}, R2 = {B, D, E, F}, R3 = {A, D, G, H, J}, R4 = {A, B, D}, R5 = {A, I}
Next, we check for transitive dependencies in each of the relations (which violate 3NF).
Only R3 has a transitive dependency {A, D} -> {H} -> {J}, so it is decomposed into R31 and R32 as follows:
R31 = {H, J}, R32 = {A, D, G, H}
The final set of 3NF relations is {R1, R2, R31, R32, R4, R5}

10.28 Solution to come

10.29 Given relation R(A,B,C,D,E) with dependencies
AB □□C
CD □□E
DE □□B
is AB a candidate key?
is ABD a candidate key?

Answers:

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

10.30 Consider the relation R, which has attributes that hold schedules of courses and sections at a university; R = {CourseNo, SecNo, OfferingDept, CreditHours, CourseLevel, InstructorSSN, Semester, Year, Days_Hours, RoomNo, NoOfStudents}. Suppose that the following functional dependencies hold on R:
{CourseNo} -> {OfferingDept, CreditHours, CourseLevel}
{CourseNo, SecNo, Semester, Year} ->
{Days_Hours, RoomNo, NoOfStudents, InstructorSSN}
{RoomNo, Days_Hours, Semester, Year} -> {InstructorSSN, CourseNo, SecNo}
Try to determine which sets of attributes form keys of R. How would you normalize this relation?

Answer:

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.

10.31 Consider the following relations for an order-processing application database at ABC, Inc.

ORDER (O#, Odate, Cust#, Total_amount)
ORDER-ITEM (O#, I#, Qty_ordered, Total_price, Discount%)

Assume that each item has a different discount. The Total_price refers to one item, Odate is the date on which the order was placed, and the Total_amount is the amount of the order. If we apply a natural join on the relations Order-Item and Order in this database, what does the resulting relation schema look like? What will be its key? Show the FDs in this resulting relation. Is it in 2NF? Is it in 3NF? Why or why not? (State any assumptions you make.)

Answer:

Given relations
Order(O#, Odate, Cust#, Total_amt)
Order_Item(O#, I#, Qty_ordered, Total_price, Discount%),
the schema of Order * Order_Item looks like
T_1(O#,I#,Odate, Cust#, Total_amount, Qty_ordered, Total_price, Discount%)
and its key is O#,I#.
It has functional dependencies
O#I# . Qty_ordered

O#I# .Total_price
O#I# .Discount%
O# . Odate
O# .Cust#
O# .Total_amount
It is not in 2NF, as attributes Odate, Cut#, and Total_amount are only partially dependent on the primary key, O#I#
Nor is it in 3NF, as a 2NF is a requirement for 3NF.

10.32 Consider the following relation:
CAR_SALE(Car#, Date_sold, Salesman#, Commision%, Discount_amt
Assume that a car may be sold by multiple salesmen and hence {CAR#, SALESMAN#} is the primary key. Additional dependencies are:
Date_sold ->Discount_amt
and
Salesman# ->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?

Answer:

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%)

10.33 Consider the following relation for published books:
BOOK (Book_title, Authorname, Book_type, Listprice, Author_affil, Publisher)
Author_affil referes to the affiliation of the author. Suppose the following dependencies exist:
Book_title -> Publisher, Book_type
Book_type -> Listprice
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.

Answer:

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 11: RELATIONAL DATABASE DESIGN ALGORITHMS AND FURTHER DEPENDENCIES

Answers to Selected Exercises

11.15 Show that the relation schemas produced by Algorithm 11.2 are in 3NF.

Answer:

We give a proof by contradiction. Suppose that one of the relations R i resulting from Algorithm 13.1 is not in 3NF. Then a FD Y -> A holds R i in where: (a) Y is not a superkey of R, and (b) A is not a prime attribute. But according to step 2 of the algorithm, R i will contain a set of attributes X union A 1 union A 2 union ... union A n , where X -> A i for i=1, 2, ..., n, implying that X is a key of R i and the A i are the only non-prime attributes of R i . Hence, if an FD Y -> A holds in R i where A is non-prime and Y is not a superkey of R i , Y must be a proper subset of X (otherwise Y would contain X and hence be a superkey). If both Y -> A and X -> A hold and Y is a proper subset of X, this contradicts that X -> A is a FD in a minimal set of FDs that is input to the algorithm, since removing an attribute from X leaves a valid FD, thus violating one of the minimality conditions. This produces a contradiction of our assumptions. Hence, R i must be in 3NF.

11.16 Show that if the matrix S resulting from Algorithm 11.1 does not have a row that is all "a" symbols, then projecting S on the decomposition and joining it back will always produce at least one spurious tuple.

Answer:

The matrix S initially has one row for each relation R i in the decomposition, with "a" symbols under the columns for the attributes in R i . Since we never change an "a" symbol into a "b" symbol during the application of the algorithm, then projecting S on each R i at the end of applying the algorithm will produce one row consisting of all "a" symbols in each S(R i). Joining these back together again will produce at least one row of all "a" symbols (resulting from joining the all "a" rows in each projection S(R i)). Hence, if after applying the algorithm, S does not have a row that is all "a", projecting S over the R i 's and joining will result in at least one all "a" row, which will be a spurious tuple (since it did not exist in S but will exist after projecting and joining over the R i 's).

11.17 Show that the relation schemas produced by Algorithm 11.3 are in BCNF.

Answer:

This is trivial, since the algorithm loop will continue to be applied until all relation schemas are in BCNF.

11.18 Show that the relation schemas produced by Algorithm 11.4 are in 3NF.

Answer:

The proof is similar to Exercise 15.15. The possible additional relation that contains a key of R will be in 3NF since all its attributes will be prime attributes.

11.19 Specify a template dependency for join dependencies.

Answer:

The following template specifies a join dependency JD(X,Y,Z).

$$R = \{ A , B , C \}$$

$$\begin{array}{l|ccc}
 & a & b & c_1 \\
\text{hypothesis} & a & b_1 & c \\
 & a_1 & b & c \\
\hline
\text{conclusion} & a & b & c
\end{array}$$

$$X = \{ A , B \}$$
$$Y = \{ B , C \}$$
$$Z = \{ A , C \}$$

11.20 Specify all the inclusion dependencies for the relational schema of Figure 5.5.

Answer:

The inclusion dependencies will correspond to the foreign keys shown in Figure 5.7.

11.21 Prove that a functional dependency is also a multivalued dependency.

Answer:

Suppose that a functional dependency X -> Y exists in a relation R={X, Y, Z}, and suppose there are two tuples with the same value of X. Because of the functional dependency, they must also have the same value of Y. Suppose the tuples are t 1 = < x, y, z 1 > and t 2 = < x, y, z 2 >. Then, according to the definition of multivalued dependency, we must have two tuples t 3 and t 4 (not necessarily distinct from t 1 and t 2) satisfying: t 3 [X]= t 4 [X]= t 1 [X]= t 2 [X], t 3 [Y]= t 2 [Y], t 4 [Y]= t 1 [Y], t 3 [Z]= t 1 [Z], and t 4 [Z]= t 2 [Z]. Two tuples satisfying this are t 2 (satisfies conditions for t 4) and t 1 (satisfies conditions for t 3). Hence, whenever the condition for functional dependency holds, so does the condition for multivalued dependency.

11.22 No solution provided.

11.23 No solution provided.

11.24 No solution provided.

11.25 No solution provided.

11.26 No solution provided.

11.27 No solution provided.

11.28 No solution provided.

11.29 No solution provided.

11.30 Consider the relation REFRIG(MODEL#, YEAR, PRICE, MANUF_PLANT, COLOR), which is abbreviated as REFRIG(M, Y, P, MP, C), and the following set of F of 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 REFRIG is in 3NF and in BCNF, giving proper reasons.

(c) Consider the decomposition of REFRIG into D={R1(M,Y,P), R2(M,MP,C)}. Is this decomposition lossless? Show why. (You may consult the test under Property LJ1 in Section 11.1.4)
Answers:

(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)

R = {M, Y, P, MP, C}
R1 = {M, Y, P}
R2 = {M, MP, C}

F = {M   MP, {M, Y}   P, MP   C}
F^+ = { $\{M\}^+$ {M, MP, C},
$\{M, Y\}^+$ {M, Y, P, MP, C},
$\{MP\}^+$ {MP, C} }

R1 ∩ R2 = M
R2 – R1 = {MP, C}

D(R1, R2} has the lossless join property since
Property: *LJ1: FD ((R1 ∩ R2)   (R2 – R1)) is in F^+*
is satisfied (due to M   {MP, C}).

CHAPTER 12: PRACTICAL DATABASE DESIGN METHODOLOGY AND USE OF UML DIAGRAMS

No exercises.

CHAPTER 13: DISK STORAGE, BASIC FILE STRUCTURES, AND HASHING

Answers to Selected Exercises

13.23 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)?

(b) How many cylinders are there?

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

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

(e) Suppose the disk drive rotates the disk pack at a speed of 2400 rpm (revolutions per minute); what is the transfer rate 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)?

(f) Suppose 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?

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

Answer:

(a) Total track size = 20 * (512+128) = 12800 bytes = 12.8 Kbytes
Useful capacity of a track = 20 * 512 = 10240 bytes = 10.24 Kbytes

(b) Number of cylinders = number of tracks = 400

(c) Total cylinder capacity = 15*2*20*(512+128) = 384000 bytes = 384 Kbytes
Useful cylinder capacity = 15 * 2 * 20 * 512 = 307200 bytes = 307.2 Kbytes

(d) Total capacity of a disk pack = 15 * 2 * 400 * 20 * (512+128)
= 153600000 bytes = 153.6 Mbytes
Useful capacity of a disk pack = 15 * 2 * 400 * 20 * 512 = 122.88 Mbytes

(e) 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) average time to locate and transfer a block = s+rd+btt = 30+12.5+1 = 43.5 msec

(g) time to transfer 20 random blocks = 20 * (s + rd + btt) = 20 * 43.5 = 870 msec
time to transfer 20 consecutive blocks using double buffering = s + rd + 20*btt
= 30 + 12.5 + (20*1) = 62.5 msec
(a more accurate estimate of the latter can be calculated using the bulk transfer rate as follows: time to transfer 20 consecutive blocks using double buffering
= s+rd+((20*B)/btr) = 30+12.5+ (10240/409.6) = 42.5+ 25 = 67.5 msec)

13.24 A file has r=20000 STUDENT records of fixed-length. Each record has the following fields: NAME (30 bytes), SSN (9 bytes), ADDRESS (40 bytes), PHONE (9 bytes), BIRTHDATE (8 bytes), SEX (1 byte), MAJORDEPTCODE (4 bytes), MINORDEPTCODE (4 bytes), CLASSCODE (4 bytes, integer), and DEGREEPROGRAM (3 bytes). An additional byte is used as a deletion marker. The file is stored on the disk whose parameters are given in Exercise 4.18.

(a) Calculate the record size R in bytes.

(b) Calculate the blocking factor bfr and the number of file blocks b assuming an unspanned organization.

(c) Calculate the average time it takes to find a record by doing a linear search on the file if (i) the file blocks are stored contiguously and double buffering is used, and (ii) the file blocks are not stored contiguously.

(d) Assume the file is ordered by SSN; calculate the time it takes to search for a record given its SSN value by doing a binary search.

Answer:

(a) R = (30 + 9 + 40 + 9 + 8 + 1 + 4 + 4 + 4 + 3) + 1 = 113 bytes

(b) bfr = floor(B / R) = floor(512 / 113) = 4 records per block
b = ceiling(r / bfr) = ceiling(20000 / 4) = 5000 blocks

(c) For linear search we search on average half the file blocks= 5000/2= 2500 blocks.
i. If the blocks are stored consecutively, and double buffering is used, the time to read 2500 consecutive blocks
= s+rd+(2500*(B/btr))= 30+12.5+(2500*(512/409.6))
= 3167.5 msec = 3.1675 sec
(a less accurate estimate is = s+rd+(2500*btt)= 30+12.5+2500*1= 2542.5 msec)
ii. If the blocks are scattered over the disk, a seek is needed for each block, so the time is: 2500 * (s + rd + btt) = 2500 * (30 + 12.5 + 1) = 108750 msec = 108.75 sec

(d) For binary search, the time to search for a record is estimated as:
ceiling(log 2 b) * (s +rd + btt)
= ceiling(log 2 5000) * (30 + 12.5 + 1) = 13 * 43.5 = 565.5 msec = 0.5655 sec

13.25 Suppose only 80% of the STUDENT records have a value for PHONE, 85% for MAJORDEPTCODE, 15% for MINORDEPTCODE, and 90% for DEGREEPROGRAM, and we use a variable-length record file. Each record has a 1-byte field type for each field occurring in the record, plus the 1-byte deletion marker and a 1-byte end-ofrecord marker. Suppose we use a spanned record organization, where each block has a 5-byte pointer to the next block (this space is not used for record storage).

(a) Calculate the average record length R in bytes.

(b) Calculate the number of blocks needed for the file.

Answer:

(a) Assuming that every field has a 1-byte field type, and that the fields not mentioned above (NAME, SSN, ADDRESS, BIRTHDATE, SEX, CLASSCODE) have values in every record, we need the following number of bytes for these fields in each record, plus 1 byte for the deletion marker, and 1 byte for the end-of-record marker:
R fixed = (30+1) + (9+1) + (40+1) + (8+1) + (1+1) + (4+1) +1+1 = 100 bytes
For the fields (PHONE, MAJORDEPTCODE, MINORDEPTCODE DEGREEPROGRAM), the average number of bytes per record is:
R variable = ((9+1)*0.8)+((4+1)*0.85)+((4+1)*0.15)+((3+1)*0.9)
= 8+4.25+0.75+3.6= 16.6 bytes
The average record size R = R fixed + R variable = 100 + 16.6 = 116.6 bytes
The total bytes needed for the whole file = r * R = 20000 * 116.6 = 2332000 bytes

(b) Using a spanned record organization with a 5-byte pointer at the end of each block, the bytes available in each block are (B-5) = (512 - 5) = 507 bytes.
The number of blocks needed for the file are:
b = ceiling((r * R) / (B - 5)) = ceiling(2332000 / 507) = 4600 blocks
(compare this with the 5000 blocks needed for fixed-length, unspanned records in Problem 4.19(b))

13.26 Suppose that a disk unit has the following parameters: seek time s=20 msec; rotational delay rd=10 msec; block transfer time btt=1 msec; block size B=2400 bytes; interblock gap size G=600 bytes. An EMPLOYEE file has the following fields: SSN, 9 bytes; LASTNAME, 20 bytes; FIRSTNAME, 20 bytes; MIDDLE INIT, 1 byte; BIRTHDATE, 10 bytes; ADDRESS, 35 bytes); PHONE, 12 bytes); SUPERVISORSSN, 9 bytes; DEPARTMENT, 4 bytes; JOBCODE, 4 bytes; deletion marker, 1 byte. The EMPLOYEE file has r=30000 STUDENT records, fixed-length format, and unspanned blocking. Write down appropriate formulas and calculate the following values for the above EMPLOYEE file:

(a) The record size R (including the deletion marker), the blocking factor bfr, and the number of disk blocks b.

(b) Calculate the wasted space in each disk block because of the unspanned organization.

(c) Calculate the transfer rate tr and the bulk transfer rate btr for this disk (see Appendix B for definitions of tr and btr).

(d) Calculate the average number of block accesses needed to search for an arbitrary record in the file, using linear search.

(e) Calculate the average time needed in msec to search for an arbitrary record in the file, using linear search, if the file blocks are stored on consecutive disk blocks and double buffering is used.

(f) Calculate the average time needed in msec to search for an arbitrary record in the file, using linear search, if the file blocks are not stored on consecutive disk blocks.

(g) Assume that the records are ordered via some key field. Calculate the average number of block accesses and the average time needed to search for an arbitrary record in the file, using binary search.

Answer:

(a) R = (9 + 20 + 20 + 1 + 10 + 35 + 12 + 9 + 4 + 4) + 1 = 125 bytes
bfr = floor(B / R) = floor(2400 / 125) = 19 records per block
b = ceiling(r / bfr) = ceiling(30000 / 19) = 1579 blocks

(b) Wasted space per block = B - (R * Bfr) = 2400 - (125 * 19) = 25 bytes

(c) Transfer rate tr= B/btt = 2400 / 1 = 2400 bytes/msec
bulk transfer rate btr= tr * (B/(B+G))
= 2400*(2400/(2400+600)) = 1920 bytes/msec

(d) For linear search we have the following cases:
i. search on key field:
if record is found, half the file blocks are searched on average: b/2= 1579/2 blocks
if record is not found, all file blocks are searched: b = 1579 blocks
ii. search on non-key field:
all file blocks must be searched: b = 1579 blocks

(e) If the blocks are stored consecutively, and double buffering is used, the time to read n consecutive blocks= s+rd+(n*(B/btr))
i. if n=b/2: time = 20+10+((1579/2)*(2400/1920))= 1016.9 msec = 1.017 sec
(a less accurate estimate is = s+rd+(n*btt)= 20+10+(1579/2)*1= 819.5 msec)
ii. if n=b: time = 20+10+(1579*(2400/1920))= 2003.75 msec = 2.004 sec
(a less accurate estimate is = s+rd+(n*btt)= 20+10+1579*1= 1609 msec)

(f) If the blocks are scattered over the disk, a seek is needed for each block, so the time to search n blocks is: n * (s + rd + btt)
i. if n=b/2: time = (1579/2)*(20+10+1)= 24474.5 msec = 24.475 sec
ii. if n=b: time = 1579*(20+10+1)= 48949 msec = 48.949 sec

(g) For binary search, the time to search for a record is estimated as:
ceiling(log 2 b) * (s +rd + btt)
= ceiling(log 2 1579) * (20+10+1) = 11 * 31 = 341 msec = 0.341 sec

13.27 A PARTS file with Part# as hash key includes records with the following Part# values: 2369, 3760, 4692, 4871, 5659, 1821, 1074, 7115, 1620, 2428, 3943, 4750, 6975, 4981, 9208. The file uses 8 buckets, numbered 0 to 7. Each bucket is one disk block and holds two records. Load these records into the file in the given order using the hash function h(K)=K mod 8. Calculate the average number of block accesses for a random retrieval on Part#.

Answer:

The records will hash to the following buckets:

K	h(K) (bucket number)
2369	1

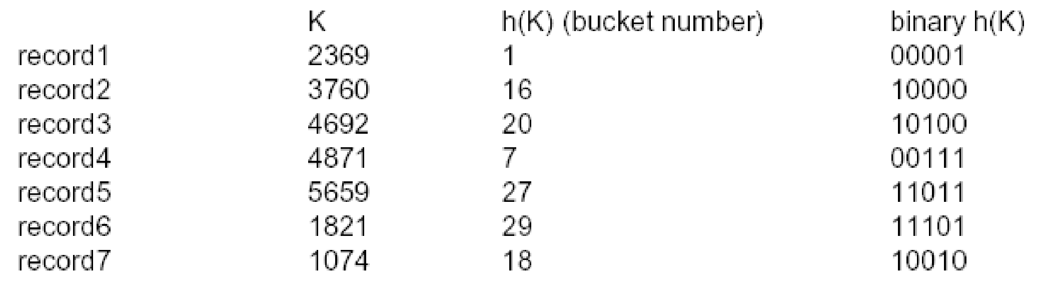
3760	0	
4692	4	
4871	7	
5659	3	
1821	5	
1074	2	
7115	3	
1620	4	
2428	4	overflow
3943	7	
4750	6	
6975	7	overflow
4981	5	
9208	0	
9209		

Two records out of 15 are in overflow, which will require an additional block access. The other records require only one block access. Hence, the average time to retrieve a random record is:
(1 * (13/15)) + (2 * (2/15)) = 0.867 + 0.266 = 1.133 block accesses

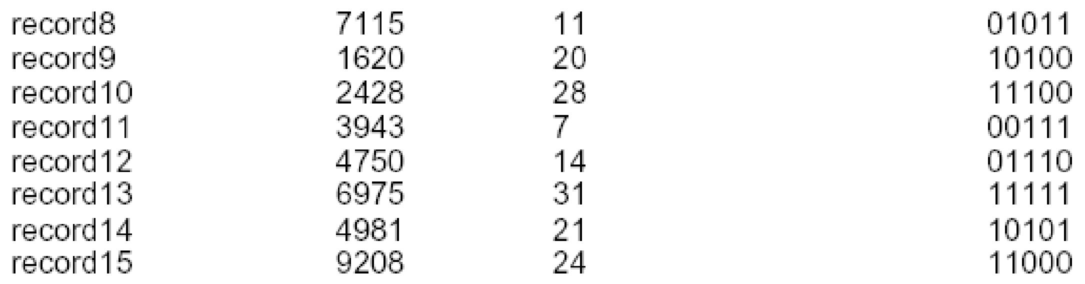
13.28 Load the records of Exercise 5.27 into expandable hash files based on extendible hashing. Show the structure of the directory at each step. Show the directory at each step, and the global and local depths. Use the has function
h(k) = K mod 32.

Answer:

Hashing the records gives the following result:

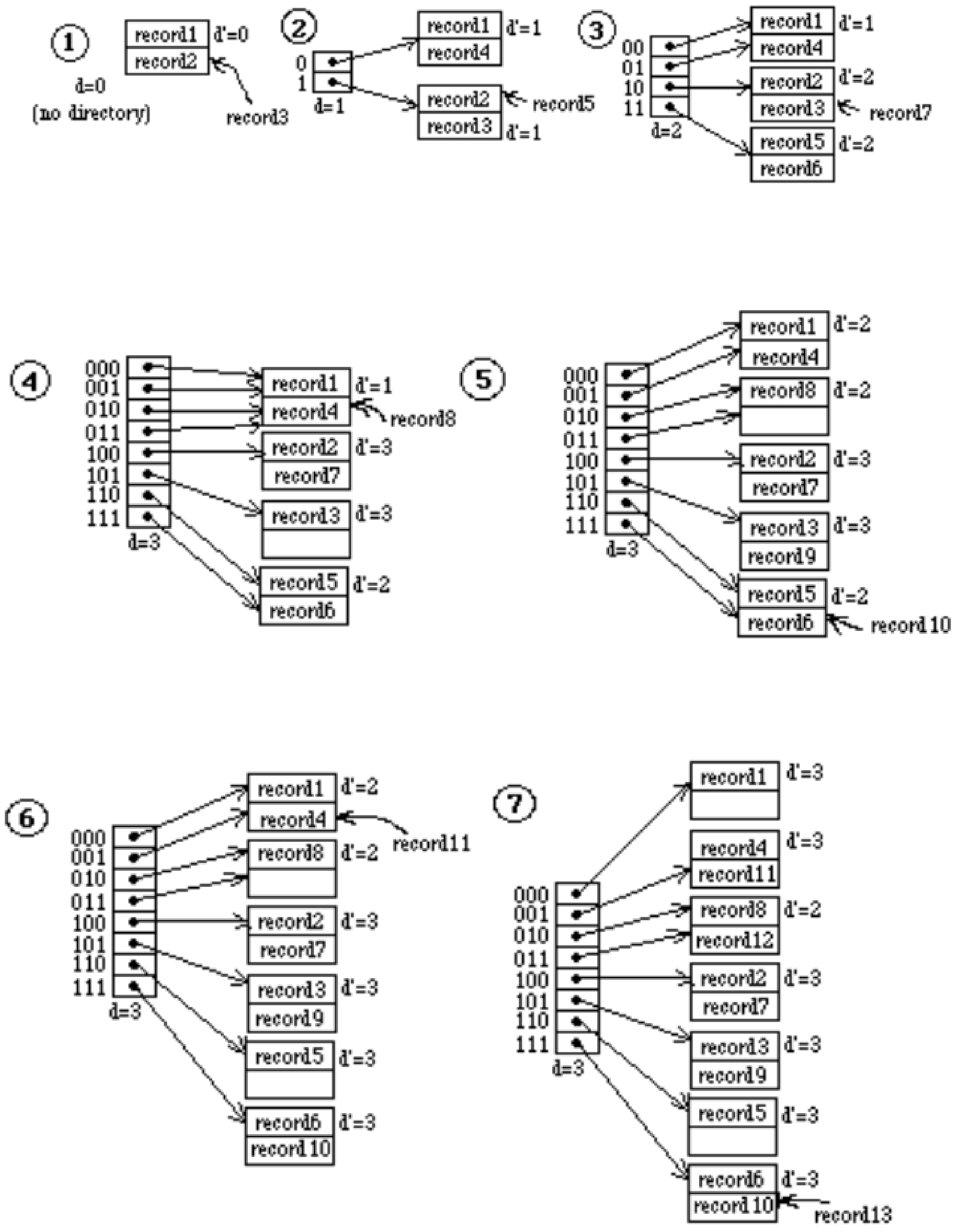


	K	h(K) (bucket number)	binary h(K)
record1	2369	1	00001
record2	3760	16	10000
record3	4692	20	10100
record4	4871	7	00111
record5	5659	27	11011
record6	1821	29	11101
record7	1074	18	10010

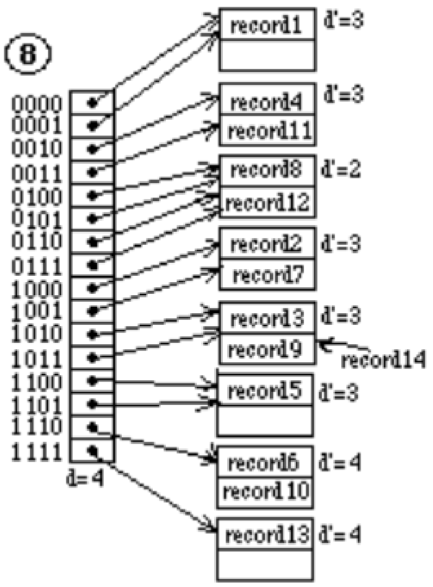


record8	7115	11	01011
record9	1620	20	10100
record10	2428	28	11100
record11	3943	7	00111
record12	4750	14	01110
record13	6975	31	11111
record14	4981	21	10101
record15	9208	24	11000

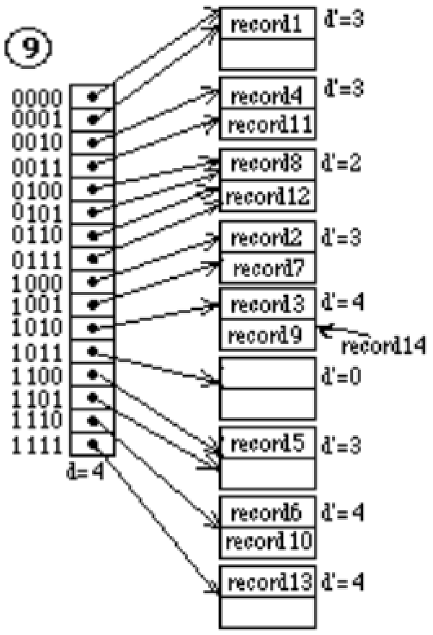
Extendible hashing:



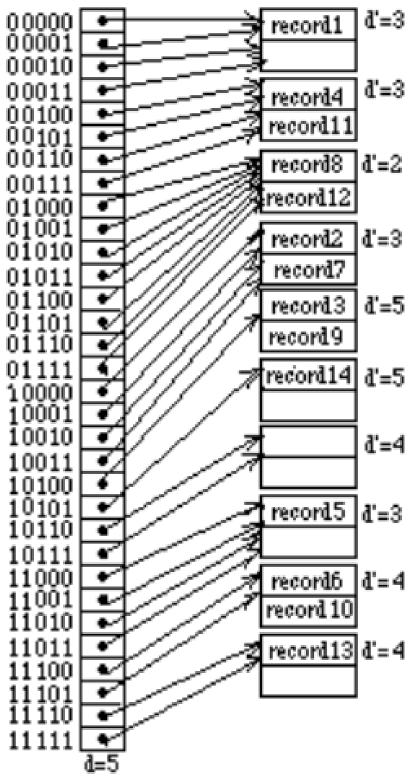
8


0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
d=4
record1
d'=3
record4
record11
d'=3
record8
record12
d'=2
record2
record7
d'=3
record3
record9
d'=3
record14
record5
d'=3
record6
record10
d'=4
record13
d'=4

9

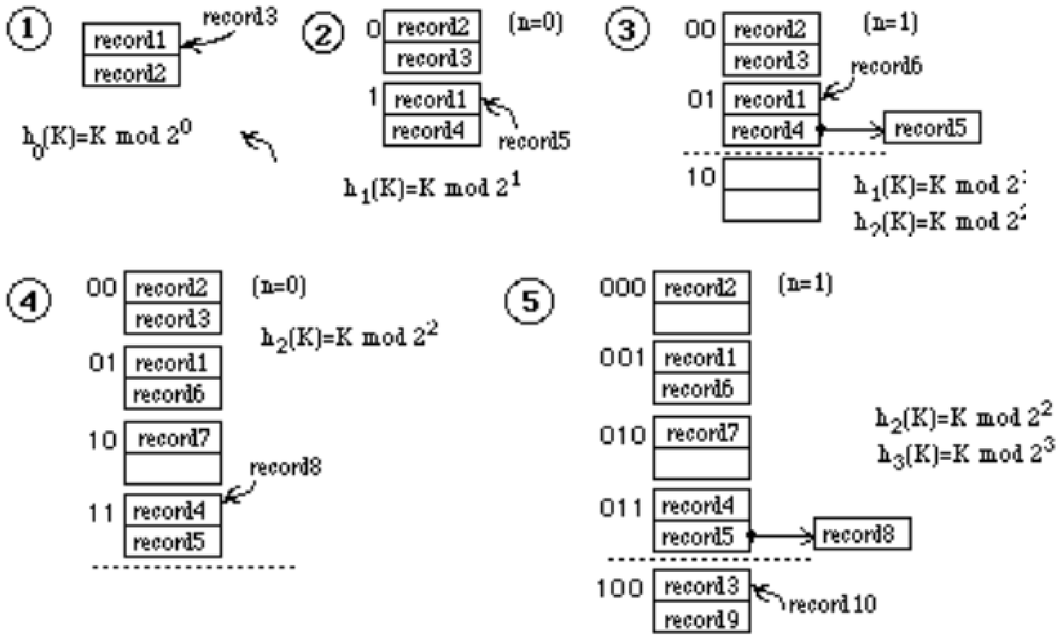

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111
d=4
record1
d'=3
record4
record11
d'=3
record8
record12
d'=2
record2
record7
d'=3
record3
record9
d'=4
record14
d'=0
record5
d'=3
record6
record10
d'=4
record13
d'=4

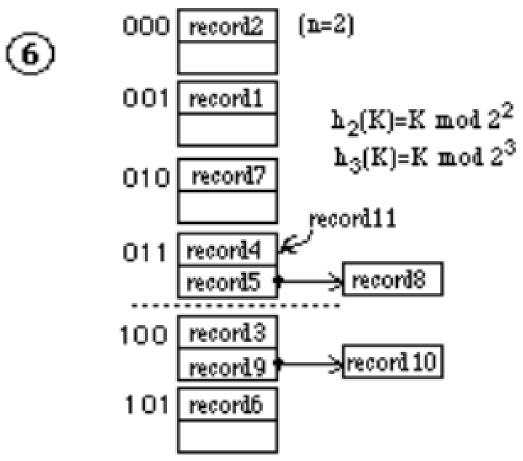
10

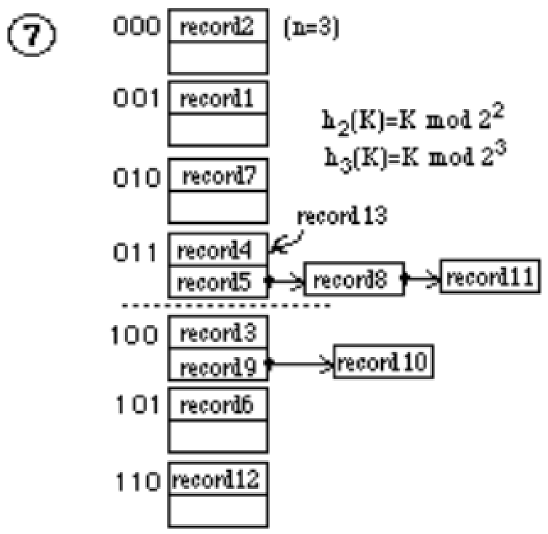

00000
00001
00010
00011
00100
00101
00110
00111
01000
01001
01010
01011
01100
01101
01110
01111
10000
10001
10010
10011
10100
10101
10110
10111
11000
11001
11010
11011
11100
11101
11110
11111
d=5
record1
d'=3
record4
record11
d'=3
record8
record12
d'=2
record2
record7
d'=3
record3
record9
d'=5
record14
d'=5
d'=4
record5
d'=3
record6
record10
d'=4
record13
d'=4

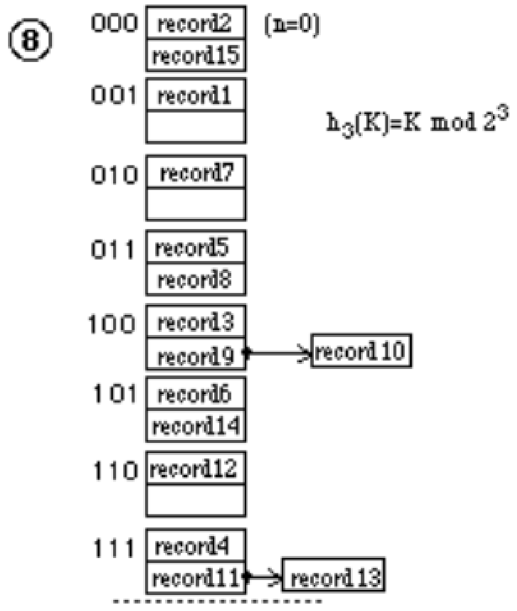
13.29 Load the records of Exercise 5.27 into expandable hash files based on linear hashing. Start with a single disk block, using the hash function *h ø = K* mod 2º, and show how the file grows and how the hash functions change as the records are inserted. Assume that blocks are split whenever an overflow occurs, and show the value of n at each stage

Answer:









Note: It is more common to specify a certain load factor for the file for triggering the splitting of buckets (rather than triggering the splitting whenever a new record being inserted is placed in overflow). The load factor lf could be defined as:
lf = (r) / (b * Bfr)

where r is the current number of records, b is the current number of buckets,
and Bfr is the maximum number of records per bucket. Whenever lf gets to be
larger than some threshold, say 0.8, a split is triggerred. It is also possible to
merge backets in the reverse order in which they were created; a merge operation
would be triggerred whenever lf becomes less than another threshold, say 0.6.

13.30 No solution provided

13.31 No solution provided

13.32 No solution provided

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

Answer:

It is possible to use an overflow file in which the records are chained together in a
manner similar to the overflow for static hash files. The overflow records that should be
inserted in each block of the ordered file are linked together in the overflow file, and a
pointer to the first record in the chain (linked list) is kept in the block of the main file.
The list may or may not be kept ordered.

13.34 No solution provided

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

Answer:

One can use techniques for handling collisions similar to those used for internal hashing.
For example, 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 (open addressing). Another scheme is to use
a whole overflow block for each bucket that is full. However, chaining seems to be the
most appropriate technique for static external hashing.

13.36 No solution provided.

13.37 No solution provided.

13.38 Suppose that a file initially contains r=120000 records of R=200 bytes each in an
unsorted (heap) file. The block size B=2400 bytes, the average seek time s=16 ms, the
average rotational latency rd=8.3 ms and the block transfer time btt=0.8 ms. Assume that 1
record is deleted for every 2 records added until the total number of active records is
240000.

(a) How many block transfers are needed to reorganize the file?

(b) How long does it take to find a record right before reorganization?
(c) How long does it take to find a record right after reorganization?

Let X = # of records deleted
Hence 2X= # of records added.

Total active records
= 240,000 = 120,000 - X + 2X.
Hence, X = 120,000
Records before reorganization (i.e., before deleting any records physically) =
360,000.

(a) No. of blocks for Reorganization
= Blocks Read + Blocks Written.
-200 bytes/record and 2400 bytes/block gives us 12 records per block
-Reading involves 360,000 records; i.e. 360,000/12 = 30K blocks
-Writing involves 240,000 records; i.e., 240000/12 = 20K blocks.
Total blocks transferred during reorganization
= 30K + 20K = 50K blocks.

(b) Time to locate a record before reorganization. On an average we assume that
half the file will be read.
Hence, Time = (b/2)* btt = 15000 * 0.8 ms = 12000 ms.
= 12 sec.

(c) Time to locate a record after reorganization
= (b/2) * btt = 10000 * 0.8 = 8 sec.

13.39 Suppose we have a sequential (ordered) file of 100000 records where each record is 240 bytes. Assume that B=2400 bytes, s=16 ms, rd=8.3 ms, and btt=0.8 ms. Suppose we want to make X independent random records from the file. We could make X random block reads or we could perform one exhaustive read of the entire file looking for those X records. The question is to decide when it would be more efficient to perform one exhaustive read of the entire file than to perform X individual random reads. That is, what is the value for X when an exhaustive read of the file is more efficient than random X reads? Develop this function of X.

Total blocks in file = 100000 records * 240 bytes/record divided by 2400
bytes/block = 10000 blocks.
Time for exhaustive read
= s + r + b.btt
= 16 + 8.3 + (10000) * 0.8
= 8024.3 msec
Let X be the # of records searched randomly that takes more time than
exhaustive read time.
Hence, X (s + r + btt) > 8024.3
X (16+8.3+0.8) > 8024.3
X > 8024.3/25.1
Thus, X > 319.69
i.e. If at least 320 random reads are to be made, it is better to search the file
exhaustively.

13.40 Suppose that a static hash file initially has 600 buckets in the primary area and that records are inserted that create an overflow area of 600 buckets. If we reorganize the hash file, we can assume that the overflow is eliminated. If the cost of reorganizing the file is the cost of the bucket transfers (reading and writing al of the buckets) and the only periodic file operation is the fetch operation, then how many times would we have to perform a fetch (successfully) to make the reorganization cost-effective? That is, the reorganization cost and subsequent search cost are less than the search cost before reorganization. Assume s=16, rd=8.3 ms, btt=1 ms.

Primary Area = 600 buckets
Secondary Area (Overflow) = 600 buckets
Total reorganization cost = Buckets Read & Buckets Written for (600 & 600) + 1200 = 2400 buckets = 2400 (1 ms) = 2400 ms
Let X = number of random fetches from the file.
Average Search time per fetch = time to access (1 + 1/2) buckets where 50% of time we need to access the overflow bucket.
Access time for one bucket access = (S + r + btt)
= 16 + 8.3 + 0-8
= 25.1 ms
Time with reorganization for the X fetches
= 2400 + X (25.1) ms
Time without reorganization for X fetches = X (25.1) (1 + 1/2) ms
= 1.5 * X * (25.1) ms.
Hence, 2400 + X (25.1) < (25.1) * (1.5X)
2374.9/ 12.55 < X
Hence, 189.23 < X
If we make at least 190 fetches, the reorganization is worthwhile.

13.41 Suppose we want to create a linear hash file with a file load factor of 0.7 and a blocking factor of 20 records per bucket, which is to contain 112000 records initially.

(a) How many buckets should we allocate in primary areas?

(b) What should be the number of bits used for bucket addresses?

Answer:

(a) No. of buckets in primary area.
= 112000/(20*0.7) = 8000.

(b) K: the number of bits used for bucket addresses
2K < = 8000 < = 2 k+1
2 12 = 4096
2 13 = 8192
K = 12
Boundary Value = 8000 - 2 12
= 8000 - 4096
= 3904 -

CHAPTER 14: INDEXING STRUCTURES FOR FILES

Answers to Selected Exercises

14.14 Consider a disk with block size B=512 bytes. A block pointer is P=6 bytes long, and a record pointer is P R =7 bytes long. A file has r=30,000 EMPLOYEE records of fixed-length. Each record has the following fields: NAME (30 bytes), SSN (9 bytes), DEPARTMENTCODE (9 bytes), ADDRESS (40 bytes), PHONE (9 bytes), BIRTHDATE (8 bytes), SEX (1 byte), JOBCODE (4 bytes), SALARY (4 bytes, real number). An additional byte is used as a deletion marker.

Answers:

(a) Calculate the record size R in bytes.

(b) Calculate the blocking factor bfr and the number of file blocks b assuming an unspanned organization.

(c) Suppose the file is ordered by the key field SSN and we want to construct a primary index on SSN. Calculate (i) the index blocking factor bfr i (which is also the index fan-out fo); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it into a multi-level index; (iv) the total number of blocks required by the multi-level index; and (v) the number of block accesses needed to search for and retrieve a record from the file--given its SSN value--using the primary index.

(d) Suppose the file is not ordered by the key field SSN and we want to construct a secondary index on SSN. Repeat the previous exercise (part c) for the secondary index and compare with the primary index.

(e) Suppose the file is not ordered by the non-key field DEPARTMENTCODE and we want to construct a secondary index on SSN using Option 3 of Section 5.1.3, with an extra level of indirection that stores record pointers. Assume there are 1000 distinct values of DEPARTMENTCODE, and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr i (which is also the index fan-out fo); (ii) the number of blocks needed by the level of indirection that stores record pointers; (iii) the number of first-level index entries and the number of first-level index blocks; (iv) the number of levels needed if we make it a multi-level index; (v) the total number of blocks required by the multi-level index and the blocks used in the extra level of indirection; and (vi) the approximate number of block accesses needed to search for and retrieve all records in the file having a specific DEPARTMENTCODE value using the index.

(f) Suppose the file is ordered by the non-key field DEPARTMENTCODE and we want to construct a clustering index on DEPARTMENTCODE that uses block anchors (every new value of DEPARTMENTCODE starts at the beginning of a new block). Assume there are 1000 distinct values of DEPARTMENTCODE, and that the EMPLOYEE records are evenly distributed among these values. Calculate (i) the index blocking factor bfr i (which is also the index fan-out fo); (ii) the number of first-level index entries and the number of first-level index blocks; (iii) the number of levels needed if we make it a multi-level index; (iv) the total number of blocks required by the multi-level index; and (v) the number of block accesses needed to search for and retrieve all records in the file having a specific DEPARTMENTCODE value using

the clustering index (assume that multiple blocks in a cluster are either contiguous
or linked by pointers).

(g) Suppose the file is not ordered by the key field SSN and we want to construct a B + -tree
access structure (index) on SSN. Calculate (i) the orders p and p leaf of the
B + -tree; (ii) the number of leaf-level blocks needed if blocks are approximately
69% full (rounded up for convenience); (iii) the number of levels needed if
internal nodes are also 69% full (rounded up for convenience); (iv) the total
number of blocks required by the B + -tree; and (v) the number of block accesses
needed to search for and retrieve a record from the file--given its SSN value--
using the B + -tree.

Answer:

(a) Record length R = (30 + 9 + 9 + 40 + 9 + 8 + 1 + 4 + 4) + 1 = 115 bytes

(b) Blocking factor bfr = floor(B/R) = floor(512/115) = 4 records per block
Number of blocks needed for file = ceiling(r/bfr) = ceiling(30000/4) = 7500

(c) i. Index record size R i = (V SSN + P) = (9 + 6) = 15 bytes
Index blocking factor bfr i = fo = floor(B/R i) = floor(512/15) = 34
ii. Number of first-level index entries r 1 = number of file blocks b = 7500 entries
Number of first-level index blocks b 1 = ceiling(r 1 /bfr i) = ceiling(7500/34)
= 221 blocks
iii. We can calculate the number of levels as follows:
Number of second-level index entries r 2 = number of first-level blocks b 1
= 221 entries
Number of second-level index blocks b 2 = ceiling(r 2 /bfr i) = ceiling(221/34)
= 7 blocks
Number of third-level index entries r 3 = number of second-level index blocks b 2
= 7 entries
Number of third-level index blocks b 3 = ceiling(r 3 /bfr i) = ceiling(7/34) = 1
Since the third level has only one block, it is the top index level.
Hence, the index has x = 3 levels
iv. Total number of blocks for the index b i = b 1 + b 2 + b 3 = 221 + 7 + 1
= 229 blocks
v. Number of block accesses to search for a record = x + 1 = 3 + 1 = 4

(d) i. Index record size R i = (V SSN + P) = (9 + 6) = 15 bytes
Index blocking factor bfr i = (fan-out) fo = floor(B/R i) = floor(512/15)
= 34 index records per block
(This has not changed from part (c) above)
(Alternative solution: The previous solution assumes that leaf-level index blocks contain
block pointers; it is also possible to assume that they contain record pointers, in
which case the index record size would be V SSN + P R = 9 + 7 = 16 bytes. In this
case, the calculations for leaf nodes in (i) below would then have to use R i = 16
bytes rather than R i = 15 bytes, so we get:
Index record size R i = (V SSN + P R) = (9 + 7) = 15 bytes
Leaf-level ndex blocking factor bfr i = floor(B/R i) = floor(512/16)
= 32 index records per block
However, for internal nodes, block pointers are always used so the fan-out for
internal nodes fo would still be 34.)

ii. Number of first-level index entries r 1 = number of file records r = 30000
Number of first-level index blocks b 1 = ceiling(r 1 /bfr i) = ceiling(30000/34)
= 883 blocks
(Alternative solution:
Number of first-level index entries r 1 = number of file records r = 30000
Number of first-level index blocks b 1 = ceiling(r 1 /bfr i) = ceiling(30000/32)
= 938 blocks)
iii. We can calculate the number of levels as follows:
Number of second-level index entries r 2 = number of first-level index blocks b 1
= 883 entries
Number of second-level index blocks b 2 = ceiling(r 2 /bfr i) = ceiling(883/34)
= 26 blocks
Number of third-level index entries r 3 = number of second-level index blocks b 2
= 26 entries
Number of third-level index blocks b 3 = ceiling(r 3 /bfr i) = ceiling(26/34) = 1
Since the third level has only one block, it is the top index level.
Hence, the index has x = 3 levels
(Alternative solution:
Number of second-level index entries r 2 = number of first-level index blocks b 1
= 938 entries
Number of second-level index blocks b 2 = ceiling(r 2 /bfr i) = ceiling(938/34)
= 28 blocks
Number of third-level index entries r 3 = number of second-level index blocks b 2
= 28 entries
Number of third-level index blocks b 3 = ceiling(r 3 /bfr i) = ceiling(28/34) = 1
Since the third level has only one block, it is the top index level.
Hence, the index has x = 3 levels)
iv. Total number of blocks for the index b i = b 1 + b 2 + b 3 = 883 + 26 + 1 = 910
(Alternative solution:
Total number of blocks for the index b i = b 1 + b 2 + b 3 = 938 + 28 + 1 = 987)
v. Number of block accesses to search for a record = x + 1 = 3 + 1 = 4

(e) i. Index record size R i = (V DEPARTMENTCODE + P) = (9 + 6) = 15 bytes
Index blocking factor bfr i = (fan-out) fo = floor(B/R i) = floor(512/15)
= 34 index records per block
ii. There are 1000 distinct values of DEPARTMENTCODE, so the average number of records for each value is (r/1000) = (30000/1000) = 30
Since a record pointer size P R = 7 bytes, the number of bytes needed at the level of indirection for each value of DEPARTMENTCODE is 7 * 30 =210 bytes, which fits in one block. Hence, 1000 blocks are needed for the level of indirection.
iii. Number of first-level index entries r 1
= number of distinct values of DEPARTMENTCODE = 1000 entries
Number of first-level index blocks b 1 = ceiling(r 1 /bfr i) = ceiling(1000/34)
= 30 blocks
iv. We can calculate the number of levels as follows:
Number of second-level index entries r 2 = number of first-level index blocks b 1
= 30 entries
Number of second-level index blocks b 2 = ceiling(r 2 /bfr i) = ceiling(30/34) = 1
Hence, the index has x = 2 levels
v. total number of blocks for the index b i = b 1 + b 2 + b indirection
= 30 + 1 + 1000 = 1031 blocks
vi. Number of block accesses to search for and retrieve the block containing the record pointers at the level of indirection = x + 1 = 2 + 1 = 3 block accesses

If we assume that the 30 records are distributed over 30 distinct blocks, we need
an additional 30 block accesses to retrieve all 30 records. Hence, total block
accesses needed on average to retrieve all the records with a given value for
DEPARTMENTCODE = x + 1 + 30 = 33

(f) i. Index record size R i = (V DEPARTMENTCODE + P) = (9 + 6) = 15 bytes
Index blocking factor bfr i = (fan-out) fo = floor(B/R i) = floor(512/15)
= 34 index records per block
ii. Number of first-level index entries r 1
= number of distinct DEPARTMENTCODE values= 1000 entries
Number of first-level index blocks b 1 = ceiling(r 1 /bfr i)
= ceiling(1000/34) = 30 blocks
iii. We can calculate the number of levels as follows:
Number of second-level index entries r 2 = number of first-level index blocks b 1
= 30 entries
Number of second-level index blocks b 2 = ceiling(r 2 /bfr i) = ceiling(30/34) = 1
Since the second level has one block, it is the top index level.
Hence, the index has x = 2 levels
iv. Total number of blocks for the index b i = b 1 + b 2 = 30 + 1 = 31 blocks
v. Number of block accesses to search for the first block in the cluster of blocks
= x + 1 = 2 + 1 = 3
The 30 records are clustered in ceiling(30/bfr) = ceiling(30/4) = 8 blocks.
Hence, total block accesses needed on average to retrieve all the records with a given
DEPARTMENTCODE = x + 8 = 2 + 8 = 10 block accesses

(g) i. For a B + -tree of order p, the following inequality must be satisfied for each
internal tree node: (p * P) + ((p - 1) * V SSN) < B, or
(p * 6) + ((p - 1) * 9) < 512, which gives 15p < 521, so p=34
For leaf nodes, assuming that record pointers are included in the leaf nodes, the
following inequality must be satisfied: (p leaf * (V SSN +P R)) + P < B, or
(p leaf * (9+7)) + 6 < 512, which gives 16p leaf < 506, so p leaf =31
ii. Assuming that nodes are 69% full on the average, the average number of key
values in a leaf node is 0.69*p leaf = 0.69*31 = 21.39. If we round this up for
convenience, we get 22 key values (and 22 record pointers) per leaf node. Since the
file has 30000 records and hence 30000 values of SSN, the number of leaf-level
nodes (blocks) needed is b 1 = ceiling(30000/22) = 1364 blocks
iii. We can calculate the number of levels as follows:
The average fan-out for the internal nodes (rounded up for convenience) is
fo = ceiling(0.69*p) = ceiling(0.69*34) = ceiling(23.46) = 24
number of second-level tree blocks b 2 = ceiling(b 1 /fo) = ceiling(1364/24)
= 57 blocks
number of third-level tree blocks b 3 = ceiling(b 2 /fo) = ceiling(57/24)= 3
number of fourth-level tree blocks b 4 = ceiling(b 3 /fo) = ceiling(3/24) = 1
Since the fourth level has only one block, the tree has x = 4 levels (counting the
leaf level). Note: We could use the formula:
x = ceiling(log fo (b 1)) + 1 = ceiling(log 24 1364) + 1 = 3 + 1 = 4 levels
iv. total number of blocks for the tree b i = b 1 + b 2 + b 3 + b 4
= 1364 + 57 + 3 + 1 = 1425 blocks
v. number of block accesses to search for a record = x + 1 = 4 + 1 = 5

14.15 A PARTS file with Part# as key field includes records with the following Part#
values: 23, 65, 37, 60, 46, 92, 48, 71, 56, 59, 18, 21, 10, 74, 78, 15, 16,

20, 24, 28, 39, 43, 47, 50, 69, 75, 8, 49, 33, 38. Suppose the search field
values are inserted in the given order in a B + -tree of order p=4 and p leaf =3;
show how the tree will expand and what the final tree looks like.

Answer:

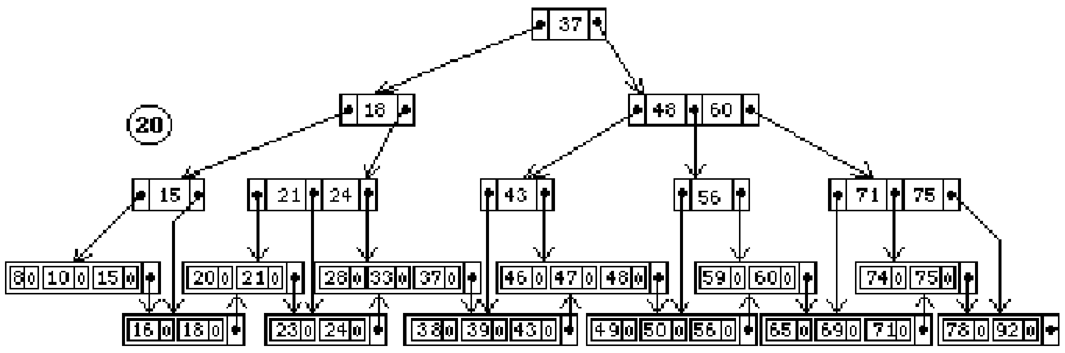
A B + -tree of order p=4 implies that each internal node in the tree (except possibly the
root) should have at least 2 keys (3 pointers) and at most 4 pointers. For p leaf =3, leaf
nodes must have at least 2 keys and at most 3 keys. The figure on page 50 shows how the
tree progresses as the keys are inserted. We will only show a new tree when insertion
causes a split of one of the leaf nodes, and then show how the split propagates up the tree.
Hence, step 1 below shows the tree after insertion of the first 3 keys 23, 65, and 37,
and before inserting 60 which causes overflow and splitting. The trees given below show
how the keys are inserted in order. Below, we give the keys inserted for each tree:
1 :23, 65, 37; 2:60; 3:46; 4:92; 5:48, 71; 6:56; 7:59, 18; 8:21; 9:10; 10:7 4 ;
11:78; 12:15; 13:16; 14:20; 15:24; 16:28, 39; 17:43, 47; 18:50, 69; 19:7 5 ;
20:8, 49, 33, 38;

14.16 No solution provided.

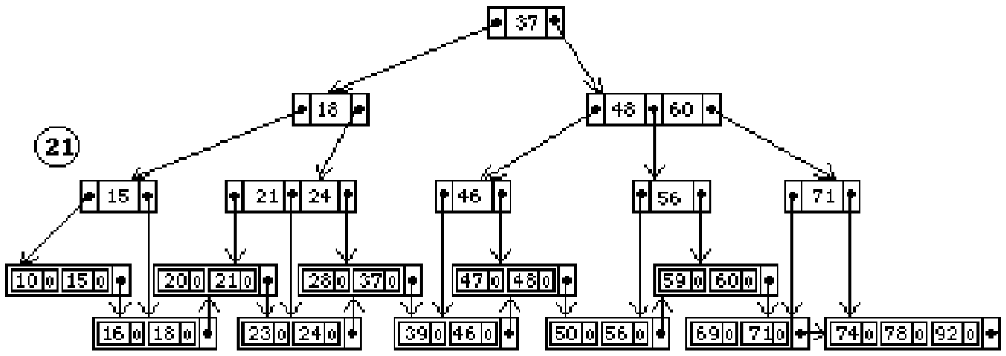
14.17 Suppose the following search field values are deleted in the given order from the
B + -tree of Exercise 6.15, show how the tree will shrink and show the final tree.
The deleted values are: 65, 75, 43, 18, 20, 92, 59, 37.

Answer:

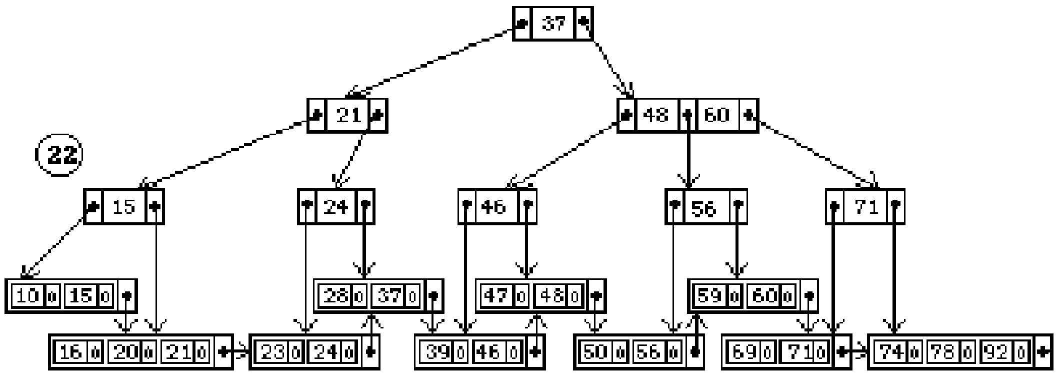
An important note about a deletion algorithm for a B + -tree is that deletion of a key value
from a leaf node will result in a reorganization of the tree if: (i) The leaf node is less
than half full; in this case, we will combine it with the next leaf node (other algorithms
combine it with either the next or the previous leaf nodes, or both), (ii) If the key value
deleted is the rightmost (last) value in the leaf node, in which case its value will appear
in an internal node; in this case, the key value to the left of the deleted key in the left
node replaces the deleted key value in the internal node. Following is what happens to the
tree number 19 after the specified deletions (not tree number 20):
Deleting 65 will only affect the leaf node. Deleting 75 will cause a leaf node to be less
than half full, so it is combined with the next node; also, 75 is removed from the
internal node leading to the following tree:



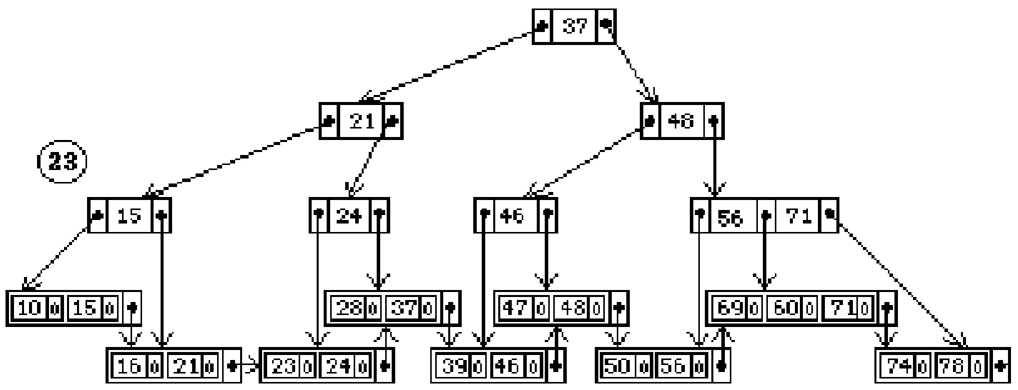
Deleting 43 causes a leaf node to be less than half full, and it is combined with the next node. Since the next node has 3 entries, its rightmost (first) entry 46 can replace 43 in both the leaf and internal nodes, leading to the following tree:



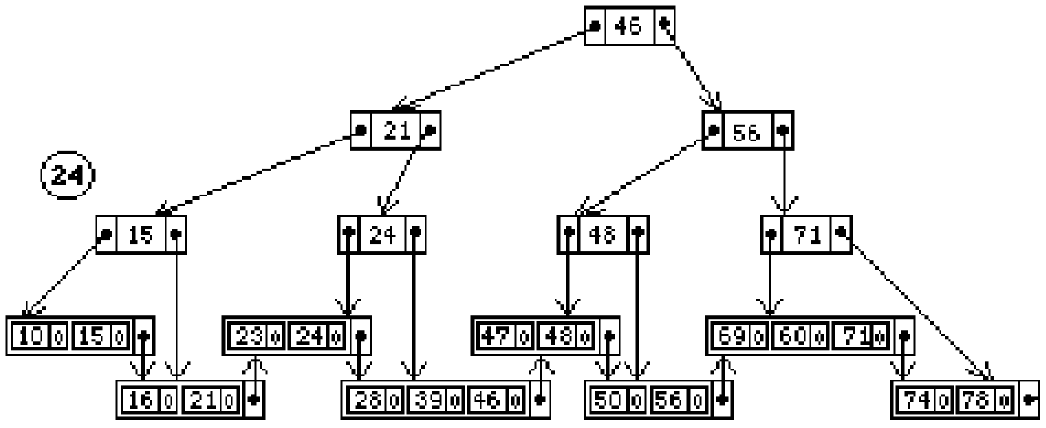
Next, we delete 18, which is a rightmost entry in a leaf node and hence appears in an internal node of the B + -tree. The leaf node is now less than half full, and is combined with the next node. The value 18 must also be removed from the internal node, causing underflow in the internal node. One approach for dealing with underflow in internal nodes is to reorganize the values of the underflow node with its child nodes, so 21 is moved up into the underflow node leading to the following tree:



Deleting 20 and 92 will not cause underflow. Deleting 59 causes underflow, and the remaining value 60 is combined with the next leaf node. Hence, 60 is no longer a rightmost entry in a leaf node and must be removed from the internal node. This is normally done by moving 56 up to replace 60 in the internal node, but since this leads to underflow in the node that used to contain 56, the nodes can be reorganized as follows:



Finally, removing 37 causes serious underflow, leding to a reorganization of the whole tree. One approach to deleting the value in the root node is to use the rightmost value in the next leaf node (the first leaf node in the right subtree) to replace the root, and move this leaf node to the left subtree. In this case, the resulting tree may look as follows:



14.18 No solution provided

14.19 No solution provided

14.20 No solution provided

14.21 No solution provided

14.22 No solution provided

14.23 No solution provided

14.24 No solution provided

CHAPTER 15: ALGORITHMS FOR QUERY PROCESSING AND OPTIMIZATION

Answers to Selected Exercises

15.13 Consider SQL queries Q1, Q8, Q1B, Q4, Q27 from Chapter 8.

(a) Draw at least two query trees that can represented each of these queries.
Under what circumstances would you use each of your query trees?

(b) Draw the initial query tree for each of these queries; then show how the
query tree is optimized by the algorithm outlined in section 18.3.2.

(c) For each query, compare your on query trees of part (a) and the initial and
final query trees of part (b).

Answer:

Below are possible answers for Q8 and Q27.
Q8: SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE E, EMPLOYEE S
WHERE E.SUPERSSN = S.SSN
Q27: SELECT FNAME, LNAME, 1.1*SALARY
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE SSN = ESSN AND PNO = PNUMBER AND PNAME = 'ProductX'
Q8's tree1:
PROJECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
E.SUPERSSN=S.SSN JOIN
EMPLOYEE E EMPLOYEE S
Q8'S tree2:
PROJECT
CARTESIAN PRODUCT
EMPLOYEE E EMPLOYEE S
E.FNAME, E.LNAME, S.FNAME, S.LNAME
SELECT E.SUPERSSN=S.SSN
The initial query tree for Q8 is the same as tree2 above; the only change made by the
optimization algorithm is to replace the selection and Cartesian product by the join in
tree1. Thus, tree 1 is the result after optimization.
Q27's tree1:
PROJECT FNAME, LNAME, SALARY
PNO=PNUMBER JOIN
EMPLOYEE PROJECT
SSN=ESSN JOIN SELECT PNAME="ProductX"
WORKS_ON
Q27's tree2:
PROJECT FNAME, LNAME, SALARY
PNO=PNUMBER AND SSN=ESSN AND PNAME="ProductX" SELECT
EMPLOYEE
PROJECT
CARTESIAN PRODUCT
WORKS_ON
CARTESIAN PRODUCT
The initial query tree of Q27 is tree2 above, but the result of the heuristic optimization
process will NOT be the same as tree1 in this case. It can be optimized more thoroughly,

as follows:
PROJECT FNAME, LNAME, SALARY
PNO=PNUMBER JOIN EMPLOYEE
PROJECT
SSN=ESSN JOIN
SELECT PNAME="ProductX" WORKS_ON
The reason is that the leaf nodes could be arranged (in Step 3 of the algorithm outlined on page 613) so that the more restrictive selects are executed first.

15.14 A file of 4096 blocks is to be sorted with an available buffer space of 64 blocks. How many passes will be needed in the merge phase of the external sort-merge algorithm?

Answer:

We first need to compute the number of runs, n , in the merge phase.
R
Using the formula in the text, we have

$$n_R = \left\lceil \frac{b}{n_B} \right\rceil$$

where $b = 4096$ (the number of blocks in the file)

$n_B = 64$ (buffer blocks)

so $n_R = 64$ (number of sorted runs on disk)

In the merge phase , the number of passes is dependent on the degree of merging, d_M

where $d_M = \text{MIN}(n_B - 1, n_R) = 63$

$$\text{The number of passes} = \left\lceil \log_{d_M} (n_R) \right\rceil = 2$$

$$= \left\lceil \log_{63} 64 \right\rceil = 2$$

15.15 Develop (approximate) cost functions for the PROJECT, UNION, INTERSECTION, SET DIFFERENCE, and CARTESIAN PRODUCT algorithms discussed in section 15.4.

Answer:

Assume relations R and S are stored in b R and b S disk blocks, respectively. Also, assume that the file resulting from the operation is stored in b RESULT disk blocks (if the size cannot be otherwise determined).
PROJECT operation: if <attribute list> includes a key of R, then the cost is 2*b R since the read–in and write–out files have the same size, which is the size of R itself; if <attribute list> does not include a key of R, then we must sort the intermediate result file before eliminating duplicates, which costs another scan; thus the latter cost is 3*b R +
k*b R *log 2 b R (assuming PROJECT is implemented by sort and eliminate duplicates).
SET OPERATIONS (UNION, DIFFERENCE, INTERSECTION): According to the algorithms where the usual sort and scan/merge of each table is done, the cost of any of these is:
k*[(b R *log 2 b R) + (b S *log 2 b S)] + b R + b S + b RESULT
CARTESIAN PRODUCT: The join selectivity of Cartesian product is js = 1, and the typical way of doing it is the nested loop method, since there are no conditions to match. We first assume two memory buffers; the other quantities are as discussed for the JOIN analysis in Section 18.2.3. We get:
J1: C R X S = b R + (b R *b S) + (|R|*|S|)/bfr RS
Next, suppose we have n B memory buffers, as in Section 16.1.2. Assume file R is smaller and is used in the outer loop. We get:
J1': C R X S = b R + ceiling(b R /(n B - 1)) * b S)+ (|R|*|S|)/bfr RS , which is better than J1, per its middle term.

15.16 No solution provided.

15.17 Can a nondense (sparse) index be used in the implementation of an aggregate operator? Why or why not?

Answer:

A nondense (sparse) index contains entries for only some of the search values. A primary index is an example of a nondense index which includes an entry for each disk block of the data file rather than for every record.

```
Index File Data File
Key ----------
_____ --------------> | 10 ... |
| | | | 12 ... |
| 10 | | | 18 ... |
| --|------ ----------
| | -----> ----------
| 22 | | | 22 ... |
| --|--------------- | 28 ... |
| | | 32 ... |
| 40 | ----------
| --|------
|____| | ----------
-------------> | 40 ... |
| 52 ... |
| 60 ... |
----------
```

If the keys in the index correspond to the smallest key value in the data block then the sparse index could be used to compute the MIN function. However, the MAX function could not be determined from the index. In addition, since all values do not appear in the index, AVG, SUM and COUNT could not be determined from just the index.

15.18 Calculate the cost functions for different options of executing the JOIN operation OP7 discussed in section 15.3.2.

Answer:

The operation is
OP7: DEPARTMENT |x| MGRSSN=SSN EMPLOYEE.
As in section 18.2.3 we assume the secondary index on MGRSSN of DEPARTMENT, with selection cardinality s=1 and level x=1; also the join selectivity of OP7 is js = 1/125 = 1/|DEPARTMENT|, because MGRSSN acts as a key of the DEPARTMENT table. (Note: There is exactly one manager per department.)
Finally, we assume the same blocking factor as the OP6 join of these two tables:
bfr = 4 records/block, as the results involve the same number of attributes. Thus the applicable methods are J1 and J2 with either table as the outer loop; the quantities parallel those of OP6:
J1 with EMPLOYEE as outer loop:
CJ1 = 2000 + (2000*13) + (((1/125)*10000*125)/4) = 30,500
J1 with DEPARTMENT as outer loop:
CJ1' = 13 + (13*2000) + (((1/125)*10000*125)/4) = 28,513
J2 with EMPLOYEE as outer loop, and MGRSSN as secondary key for S:
[EMPLOYEE as outer loop and primary index on SSN gives the same result.]
CJ2a = b R + (|R|*(x S + s)) + ((js*|R|*|S|)/bfr
= 2000 + (2000*(1+1)) + (((1/125)*10000*125)/4)
= 24,500
J2 with DEPARTMENT as outer loop:
CJ2c = b S + (|S|*(x R + 1)) + ((js*|R|*|S|)/bfr
= 13 + (125*2) + (((1/125)*10000*125)/4)
= 13 + 250 + 2500 = 2,763 [!]
Obviously this optimization was worthwhile, to get the latter minimum.

15.19 No solution provided.

15.20 No solution provided.

15.21 Extend the sort-merge join algorithm to implement the left outer join.

Answer:

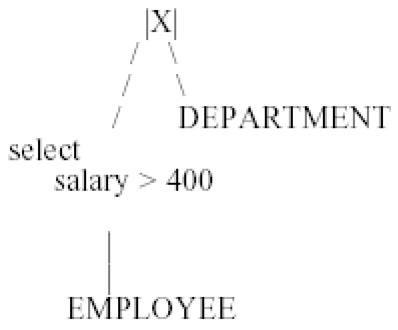
The left outer join of R and S would produce the rows from R and S that join as well as the rows from R that do not join any row from S. The sort-merge join algorithm would only need minor modifications during the merge phase. During the first step, relation R would be sorted on the join column(s).
During the second step, relation S would be sorted on the join columns(s).
During the third step, the sorted R relation and S relation would be merged. That is, rows would be combined if the R row and S row have the same value for the join column(s).
In addition, if no matching S row was found for an R row, then that R row would also be placed in the left outer join result, except that the corresponding attributes from the S relation would be set to NULL values. (How a null value is to be represented is a matter of choice out of the options provided in a DBMS.)

15.22 Compare the cost of two different query plans for the following query:

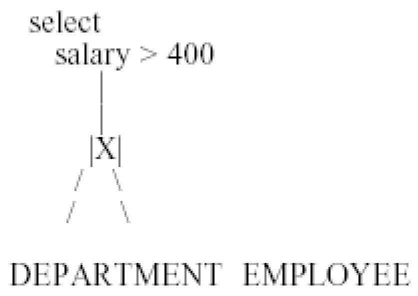
salary > 40000 select (EMPLOYEE |X| DNO=DNUMBER DEPARTMENT)
Use the database statistics in Figure 15.8

Answer:

One plan might be for the following query tree



We can use the salary index on EMPLOYEE for the select operation:
The table in Figure 18.8(a) indicates that there are 500 unique salary values, with a low value of 1 and a high value of 500. (It might be in reality that salary is in units of 1000 dollars, so 1 represents $1000 and 500 represents $500,000.)
The selectivity for (Salary > 400) can be estimated as
(500 - 400)/500 = 1/5
This assumes that salaries are spread evenly across employees.
So the cost (in block accesses) of accessing the index would be
Blevel + (1/5) * (LEAF BLOCKS) = 1 + (1/5)* 50 = 11
Since the index is nonunique, the employees can be stored on any of the data blocks.
So the the number of data blocks to be accessed would be
(1/5) * (NUM_ROWS) = (1/5) * 10,000 = 2000
Since 10,000 rows are stored in 2000 blocks, we have that
2000 rows can be stored in 400 blocks. So the TEMPORARY table (i.e., the result of the selection operator) would contain 400 blocks.
The cost of writing the TEMPORARY table to disk would be
400 blocks.
Now, we can do a nested loop join of the temporary table and the DEPARTMENT table. The cost of this would be
b + (b * b)
DEPARTMENT DEPARTMENT TEMPORARY
We can ignore the cost of writing the result for this comparision, since the cost would be the same for both plans, we will consider.
We have 5 + (5 * 400) = 2005 block accesses
18.22 (continued)
Therefore, the total cost would be
11 + 2000 + 400 + 2005 = 4416 block accesses
NOTE: If we have 5 main memory buffer pages available during the join, then we could store all 5 blocks of the DEPARTMENT table there. This would reduce the cost of the join to 5 + 400 = 405 and the total cost would be reduced to 11 + 2000 + 400 + 405 = 2816. A second plan might be for the following query tree



Again, we could use a nested loop for the join but instead of creating a temporary table for the result we can use a pipelining approach and pass the joining rows to the select operator as they are computed. Using a nested loop join algorithm would yield the following 50 + (50 * 2000) = 100,050 blocks We would pipeline the result to the selection operator and it would choose only those rows whose salary value was greater than 400.
NOTE: If we have 50 main memory buffer pages available during the join, then we could store the entire DEPARTMENT table there. This would reduce the cost of the join and the pipelined select to 50 + 2000 = 2050.

CHAPTER 16: PRACTICAL DATABASE DESIGN AND TUNING

No exercises.

CHAPTER 17: INTRODUCTION TO TRANSACTION PROCESSING CONCEPTS AND THEORY

Answers to Selected Exercises

17.14 Change transaction T 2 in Figure 17.2b to read:

```
read_item(X);
X:= X+M;
if X > 90 then exit
else write_item(X);
```

Discuss the final result of the different schedules in Figure 17.3 where M = 2
and N = 2, with respect to the following questions. Does adding the above condition
change the final outcome? Does the outcome obey the implied consistency rule
(that the capacity of X is 90)?

Answer:

The above condition does not change the final outcome unless the initial value of X > 88.
The outcome, however, does obey the implied consistency rule that X < 90, since the
value of X is not updated if it becomes greater than 90.

17.15 Repeat Exercise 19.14 adding a check in T 1 so that Y does not exceed 90.

Answer:

```
T1 T2
read_item(X);
X := X-N;
read_item(X);
X := X+M;
write_item(X);
read_item(Y);
if X > 90 then exit
else write_item(X);
Y := Y+N;
if Y> 90 then
exit
else write_item(Y);
```

The above condition does not change the final outcome unless the initial value of X > 88 or
Y > 88. The outcome obeys the implied consistency rule that X < 90 and Y < 90.

17.16 Add the operation commit at the end of each of the transactions T 1 and T 2 from
Figure 17.2; then list all possible schedules for the modified transactions.
Determine which of the schedules are recoverable, which are cascadeless, and
which are strict.

Answer:

```
T 1 T 2
read_item(X); read_item(X);
X := X - N X := X + M;
write_item(X); write_item(X);
read_item(Y); commit T 2
```

Y := Y + N;
write_item(Y);
commit T 1
The transactions can be written as follows using the shorthand notation:
T 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ;
T 2 : r 2 (X); w 2 (X); C 2 ;
In general, given m transactions with number of operations n1, n2, ..., nm, the number of possible schedules is: (n1 + n2 + ... + nm)! / (n1! * n2! * ... * nm!), where ! is the factorial function. In our case, m =2 and n1 = 5 and n2 = 3, so the number of possible schedules is:
(5+3)! / (5! * 3!) = 8*7*6*5*4*3*2*1/ 5*4*3*2*1*3*2*1 = 56.
Below are the 56 possible schedules, and the type of each schedule:
S 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; r 2 (X); w 2 (X); C 2 ; strict (and hence cascadeless)
S 2 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); C 1 ; w 2 (X); C 2 ; recoverable
S 3 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); C 1 ; C 2 ; recoverable
S 4 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); C 2 ; C 1 ; non-recoverable
S 5 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); C 1 ; w 2 (X); C 2 ; recoverable
S 6 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); w 2 (X); C 1 ; C 2 ; recoverable
S 7 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); w 2 (X); C 2 ; C 1 ; non-recoverable
S 8 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); C 1 ; C 2 ; recoverable
S 9 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); C 2 ; C 1 ; non-recoverable
S 10 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); C 2 ; w 1 (Y); C 1 ; non-recoverable
S 11 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); C 1 ; w 2 (X); C 2 ; recoverable
S 12 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); w 2 (X); C 1 ; C 2 ; recoverable
S 13 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); w 2 (X); C 2 ; C 1 ; non-recoverable
S 14 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 2 (X); w 1 (Y); C 1 ; C 2 ; recoverable
S 15 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 2 (X); w 1 (Y); C 2 ; C 1 ; non-recoverable
S 16 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 2 (X); C 2 ; w 1 (Y); C 1 ; non-recoverable
S 17 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; recoverable
S 18 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; non-recoverable
S 19 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; non-recoverable
S 20 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; non-recoverable
S 21 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; w 2 (X); C 2 ; strict (and hence cascadeless)
S 22 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 1 ; C 2 ; cascadeless
S 23 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 2 ; C 1 ; cascadeless
S 24 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); C 1 ; C 2 ; cascadeless
S 25 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); C 2 ; C 1 ; cascadeless
S 26 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); C 2 ; w 1 (Y); C 1 ; cascadeless
S 27 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; cascadeless
S 28 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; cascadeless
S 29 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; cascadeless
S 30 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 31 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; cascadeless
S 32 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; cascadeless
S 33 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; cascadeless
S 34 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 35 : r 1 (X); r 2 (X); w 2 (X); C 2 ; w 1 (X); r 1 (Y); w 1 (Y); C 1 ; strict (and hence cascadeless)
S 36 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; w 2 (X); C 2 ; strict (and hence cascadeless)
S 37 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 1 ; C 2 ; cascadeless

S 38 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); C 2 ; C 1 ; cascadeless
S 39 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); C 1 ; C 2 ; cascadeless
S 40 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); C 2 ; C 1 ; cascadeless
S 41 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 2 (X); C 2 ; w 1 (Y); C 1 ; cascadeless
S 42 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; cascadeless
S 43 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; cascadeless
S 44 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; cascadeless
S 45 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 46 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; cascadeless
S 47 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; cascadeless
S 48 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; cascadeless
S 49 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 50 : r 2 (X); r 1 (X); w 2 (X); C 2 ; w 1 (X); r 1 (Y); w 1 (Y); C 1 ; cascadeless
S 51 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; C 2 ; non-recoverable
S 52 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 2 ; C 1 ; recoverable
S 53 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); r 1 (Y); C 2 ; w 1 (Y); C 1 ; recoverable
S 54 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); C 2 ; r 1 (Y); w 1 (Y); C 1 ; recoverable
S 55 : r 2 (X); w 2 (X); r 1 (X); C 2 ; w 1 (X); r 1 (Y); w 1 (Y); C 1 ; recoverable
S 56 : r 2 (X); w 2 (X); C 2 ; r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); C 1 ; strict (and hence cascadeless)

17.17 List all possible schedules for transactions T 1 and T 2 from figure 17.2, and determine which are conflict serializable (correct) and which are not.

Answer:

T 1 T 2
read_item(X); read_item(X);
X := X - N X := X + M;
write_item(X); write_item(X);
read_item(Y);
Y := Y + N;
write_item(Y);
The transactions can be written as follows using shorthand notation:
T 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y);
T 2 : r 2 (X); w 2 (X);
In this case, m =2 and n1 = 4 and n2 = 2, so the number of possible schedules is:
(4+2)! / (4! * 2!) = 6*5*4*3*2*1/ 4*3*2*1*2*1 = 15.
Below are the 15 possible schedules, and the type of each schedule:
S 1 : r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); r 2 (X); w 2 (X); serial (and hence also serializable)
S 2 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 1 (Y); w 2 (X); (conflict) serializable
S 3 : r 1 (X); w 1 (X); r 1 (Y); r 2 (X); w 2 (X); w 1 (Y); (conflict) serializable
S 4 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 1 (Y); w 2 (X); (conflict) serializable
S 5 : r 1 (X); w 1 (X); r 2 (X); r 1 (Y); w 2 (X); w 1 (Y); (conflict) serializable
S 6 : r 1 (X); w 1 (X); r 2 (X); w 2 (X); r 1 (Y); w 1 (Y); (conflict) serializable
S 7 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); not (conflict) serializable
S 8 : r 1 (X); r 2 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); not (conflict) serializable
S 9 : r 1 (X); r 2 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); not (conflict) serializable
S 10 : r 1 (X); r 2 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); not (conflict) serializable
S 11 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); w 2 (X); not (conflict) serializable
S 12 : r 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 2 (X); w 1 (Y); not (conflict) serializable
S 13 : r 2 (X); r 1 (X); w 1 (X); w 2 (X); r 1 (Y); w 1 (Y); not (conflict) serializable
S 14 : r 2 (X); r 1 (X); w 2 (X); w 1 (X); r 1 (Y); w 1 (Y); not (conflict) serializable

S 15 : r 2 (X); w 2 (X); r 1 (X); w 1 (X); r 1 (Y); w 1 (Y); serial (and hence also serializable)

17.18 How many serial schedules exist for the three transactions in Figure 17.8 (a)? What are they? What is the total number of possible schedules?

Partial Answer:

T1 T2 T3 T3 T2 T1
T2 T3 T1 T2 T1 T3
T3 T1 T2 T1 T3 T2
Total number of serial schedules for the three transactions = 6
In general, the number of serial schedules for n transactions is n! (i.e. factorial(n))

17.19 No solution provided.

17.20 Why is an explicit transaction end statement needed in SQL but not an explicit begin statement?

Answer:

A transaction is an atomic operation. It has only one way to begin, that is with "Begin Transaction" command but it could end up in two ways: Successfully installs its updates to the database (i.e., commit) or Removes its partial updates (which may be incorrect) from the database (abort). Thus, it is important for the database systems to identify the right way of ending a transaction. It is for this reason an "End" command is needed in SQL2 query.

17.21 Describe situations where each of the different isolation levels would be useful for transaction processing.

Answer:

Transaction isolation levels provide a measure of the influence of other concurrent transactions on a given transaction. This affects the level of concurrency, that is, the level of concurrency is the highest in Read Uncommitted and the lowest in Serializable.
Isolation level Serializable: This isolation level preserves consistency in all situations, thus it is the safest execution mode. It is recommended for execution environment where every update is crucial for a correct result. For example, airline reservation, debit credit, salary increase, and so on.
Isolation level Repeatable Read: This isolation level is similar to Serializable except Phantom problem may occur here. Thus, in record locking (finer granularity), this isolation level must be avoided. It can be used in all types of environments, except in the environment where accurate summary information (e.g., computing total sum of a all different types of account of a bank customer) is desired.
Isolation level Read Committed: In this isolation level a transaction may see two different values of the same data items during its execution life. A transaction in this level applies write lock and keeps it until it commits. It also applies a read (shared) lock but the lock is released as soon as the data item is read by the transaction. This isolation level may be used for making balance, weather, departure or arrival times, and so on.
Isolation level Read Uncommitted: In this isolation level a transaction does not either apply a shared lock or a write lock. The transaction is not allowed to write any data item, thus it may give rise to dirty read, unrepeatable read, and phantom. It may be used in the environment where statistical average of a large number of data is required.

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

(a) r1 (X); r3 (X); w1(X); r2(X); w3(X)

(b) r1 (X); r3 (X); w3(X); w1(X); r2(X)

(c) r3 (X); r2 (X); w3(X); r1(X); w1(X)

(d) r3 (X); r2 (X); r1(X); w3(X); w1(X)

Answer:

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).

(a) This schedule is not serializable because T1 reads X (r1(X)) before T3 but T3 reads X (r3(X)) before T1 writes X (w1(X)), where X is a common data item. 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, T2, and T3, the operation w1(X) comes after r3(X), which does not happen in the question.

(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.

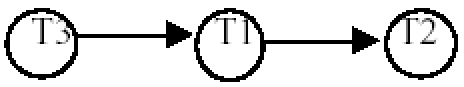
(d) This is not a serializable schedule because T3 reads X (r3(X)) before T1 reads X (r1(X)) but r1(X) happens before T3 writes X (w3(X)). In a serial schedule T3, T2, and T1, r1(X) will happen after w3(X), which does not happen in the question.

17.23 Consider the three transactions T1, T2, and T3, and the schedules S1 and S2 given below. Draw the serializibility (precedence) graphs for S1 and S2 and state whether each schedule is serializable or not. If a schedule is serializable, write down the equivalent serial schedule(s).
T1: r1(x); r1(z); w1(x)
T2: r2(z); r2(y); w2(z); w2(y)
T3: r3(x); r3(y); w3(y)
S1: r1(x); r2(z); r1(x); r3(x); r3(y); w1(x); w3(y); r2(y); w2(z); w2(y)
S2: r1(x); r2(z); r3(x); r1(z); r2(y); r3(y); w1(x); w2(z); w3(y); w2(y)

Answers:

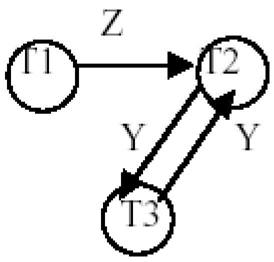
Schedule S1: It is a serializable schedule because

- □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.



17.24 Consider schedules S3, S4, and S5 below. Determine whether each schedule is strict, cascadeless, recoverable, or nonrecoverable. (Determine the strictest recoverability condition that each schedule satisfies.)
S3: r1(x); r2(z); r1(z); r3(x); r3(y); w1(x); c1; w3(y); c3; r2(y); w2(z); w2(y);c2
S4: r1(x); r2(z); r1(z); r3(x); r3(y); w1(x); w3(y); r2(y); w2(z); w2(y); c1; c2; c3;
S5: r1(x); r2(z); r3(x); r1(z); r2(y); r3(y); w1(x); w2(z); w3(y); w2(y); c3; c2;

Answer:

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-existentvalue 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.

CHAPTER 18: CONCURRENCY CONTROL TECHNIQUES

Answers to Selected Exercises

18.20 Prove that the basic two-phase locking protocol guarantees conflict serializability of schedules. (Hint: Show that, if a serializability graph for a schedule has a cycle, then at least one of the transactions participating in the schedule does not obey the two-phase locking protocol.)

Answer:

(This proof is by contradiction, and assumes binary locks for simplicity. A similar proof can be made for shared/exclusive locks.)
Suppose we have n transactions T1, T2, ..., Tn such that they all obey the basic two-phase locking rule (i.e. no transaction has an unlock operation followed by a lock operation). Suppose that a non-(conflict)-serializable schedule S for T1, T2, ..., Tn does occur; then, according to Section 17.5.2, the precedence (serialization) graph for S must have a cycle. Hence, there must be some sequence within the schedule of the form:
S: ...; [o1(X); ...; o2(X);] ...; [o2(Y); ...; o3(Y);] ... ; [on(Z); ...; o1(Z);]...
where each pair of operations between square brackets [o,o] are conflicting (either [w, w], or [w, r], or [r,w]) in order to create an arc in the serialization graph. This implies that in transaction T1, a sequence of the following form occurs:
T1: ...; o1(X); ... ; o1(Z); ...
Furthermore, T1 has to unlock item X (so T2 can lock it before applying o2(X) to follow the rules of locking) and has to lock item Z (before applying o1(Z), but this must occur after Tn has unlocked it). Hence, a sequence in T1 of the following form occurs:
T1: ...; o1(X); ...; unlock(X); ... ; lock(Z); ...; o1(Z); ...
This implies that T1 does not obey the two-phase locking protocol (since lock(Z) follows unlock(X)), contradicting our assumption that all transactions in S follow the two-phase locking protocol.

18.21 Modify the data structures for multiple-mode locks and the algorithms for read_lock(X), write_lock(X), and unlock(X) so that upgrading and downgrading of locks are possible. (Hint: The lock needs to keep track of the transaction id(s) that hold the lock, if any.)

Answer:

We assume that a List of transaction ids that have read-locked an item is maintained, as well as the (single) transaction id that has write-locked an item. Only read_lock and write_lock are shown below.

```
read_lock (X, Tn):
B: if lock (X) = "unlocked"
then begin lock (X) <- "read_locked, List(Tn)";
no_of_reads (X) <- 1
end
else if lock(X) = "read_locked, List"
then begin
(* add Tn to the list of transactions that have read_lock on X *)
lock (X) <- "read_locked, Append(List,Tn)";
no_of_reads (X) <- no_of_reads (X) + 1
end
else if lock (X) = "write_locked, Tn"
```

```
(* downgrade the lock if write_lock on X is held by Tn itself *)
then begin lock (X) <- "read_locked, List(Tn)";
no_of_reads (X) <- 1
end
else begin
wait (until lock (X) = "unlocked" and
the lock manager wakes up the transaction);
goto B;
end;
write_lock (X,Tn);
B: if lock (X) = "unlocked"
then lock (X) <- "write_locked, Tn"
else
if ( (lock (X) = "read_locked, List") and (no_of_reads (X) = 1)
and (transaction in List = Tn) )
(* upgrade the lock if read_lock on X is held only by Tn itself *)
then lock (X) = "write_locked, Tn"
else begin
wait (until ( [ lock (X) = "unlocked" ] or
[ (lock (X) = "read_locked, List") and (no_of_reads (X) = 1)
and (transaction in List = Tn) ] ) and
the lock manager wakes up the transaction);
goto B;
end;
```

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

Answer:

Since no other transaction can read or write an item written by a transaction T until T has committed, the condition for a strict schedule is satisfied.

18.23 Solution pending.

18.24 Prove that cautious waiting avoids deadlock.

Answer:

In cautious waiting, a transaction Ti can wait on a transaction Tj (and hence Ti becomes blocked) only if Tj is not blocked at that time, say time b(Ti), when Ti waits. Later, at some time b(Tj) > b(Ti), Tj can be blocked and wait on another transaction Tk only if Tk is not blocked at that time. However, Tj cannot be blocked by waiting on an already blocked transaction since this is not allowed by the protocol. Hence, the wait-for graph among the blocked transactions in this system will follow the blocking times and will never have a cycle, and so deadlock cannot occur.

18.25 Apply the timestamp ordering algorithm to the schedules of Figure 17.8(b) and (c), and determine whether the algorithm will allow the execution of the schedules.

Answer:

Let us assume a clock with linear time points 0, 1, 2, 3, ..., and that the original read and write timestamps of all items are 0 (without loss of generality).

read_TS(X) = read_TS(Y) = read_TS(Z) = 0
write_TS(X) = write_TS(Y) = write_TS(Z) = 0
Let us call the schedules in Figure 17.8(b) Schedule E or SE, and that in Figure 17.8(c)
Schedule F or SF. The two schedules can be written as follows in shorthand notation:
SE:
r2(Z); r2(Y); w2(Y); r3(Y); r3(Z); r1(X); w1(X); w3(Y); w3(Z); r2(X); r1(Y); w1(Y);
w2(X);
1 2 3 4 5 6 7 8 9 10 11 12 13
SF:
r3(Y); r3(Z); r1(X); w1(X); w3(Y); w3(Z); r2(Z); r1(Y); w1(Y); r2(Y); w2(Y); r2(X);
w2(X);
1 2 3 4 5 6 7 8 9 10 11 12 13
Assume that each operation takes one time unit, so that the numbers under the operations
indicate the time when each operation occurred. Also assume that each transaction
timestamp corresponds to the time of its first operations in each schedule, so the
transaction timestamps are as follows (Note: These values do not change during the
schedule, since they are assigned as unique identifiers to the transactions):
Schedule E Schedule F
TS(T1) = 6 TS(T1) = 3
TS(T2) = 1 TS(T2) = 7
TS(T3) = 4 TS(T3) = 1

(a) Applying timestamp ordering to Schedule E:
Initial values (new values are shown after each operation):
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
TS(T1)=6, TS(T2)=1, TS(T3)=4 (These do not change)
T2: read_item(Z)
TS(T2) > write_TS(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T2: read_item(Y)
TS(T2) > write_TS(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T2: write_item(Y)
TS(T2) = read_TS(Y) and TS(T2) > write_TS(Y)
Execute write_item(Y)
write_TS(Y) <- max(write_TS(Y),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=1,write_TS(Z)=0
T3: read_item(Y)
TS(T3) > write_TS(Y)
Execute read_item(Y)
read_TS(Y) <- max(read_TS(Y),TS(T3)) = 4
read_TS(X)=0,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=1,write_TS(Z)=0
T3: read_item(Z)
TS(T3) > write_TS(Z)
Execute read_item(Z)
read_TS(Z) <- max(read_TS(Z),TS(T3)) = 4
read_TS(X)=0,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=1,write_TS(Z)=0
T1: read_item(X)
TS(T1) > write_TS(X)

Execute read_item(X)
read_TS(X) <- max(read_TS(X),TS(T1)) = 6
read_TS(X)=6,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=1,write_TS(Z)=0
T1: write_item(X)
TS(T1) = read_TS(X) and TS(T1) > write_TS(X)
Execute write_item(X)
write_TS(X) <- max(write_TS(X),TS(T1)) = 6
read_TS(X)=6,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=6,write_TS(Y)=1,write_TS(Z)=0
T3: write_item(Y)
TS(T3) = read_TS(Y) and TS(T3) > write_TS(Y)
Execute write_item(Y)
write_TS(Y) <- max(write_TS(Y),TS(T3)) = 4
read_TS(X)=6,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=6,write_TS(Y)=4,write_TS(Z)=0
T3: write_item(Z)
TS(T3) > read_TS(Z) and TS(T3) > write_TS(Z)
Execute write_item(Z)
write_TS(Z) <- max(write_TS(Z),TS(T3)) = 4
read_TS(X)=6,read_TS(Y)=4,read_TS(Z)=1,write_TS(X)=6,write_TS(Y)=4,write_TS(Z)=4
T2: read_item(X)
TS(T2) < write_TS(X)
Abort and Rollback Y2, Reject read_item(X)

Result: Since T3 had read the value of Y that was written by T2, T3 should also be aborted and rolled by the recovery technique (because of cascading rollback); hence, all effects of T2 and T3 would also be erased and only T1 would finish execution.

(b) Applying timestamp ordering to Schedule F:
Initial values (new values are shown after each operation):
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
TS(T1)=3, TS(T2)=7, TS(T3)=1 (These do not change)
T3: read_item(Y)
TS(T3) > write_TS(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T3)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T3: read_item(Z)
TS(T3) > write_TS(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T3)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T1: read_item(X)
TS(T1) > write_TS(X)
Execute read_item(X)
read_TS(X) <- max(read_TS(X),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T1: write_item(X)
TS(T1) = read_TS(X) and TS(T1) > write_TS(X)
Execute write_item(X)
write_TS(X) <- max(write_TS(X),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=3,write_TS(Y)=0,write_TS(Z)=0
T3: write_item(Y)
TS(T3) = read_TS(Y) and TS(T3) > write_TS(Y)
Execute write_item(Y)

write_TS(Y) <- max(write_TS(Y),TS(T3)) = 1
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=3,write_TS(Y)=1,write_TS(Z)=0
T3: write_item(Z)
TS(T3) = read_TS(Z) and TS(T3) > write_TS(Z)
Execute write_item(Z)
write_TS(Z) <- max(write_TS(Z),TS(T3)) = 1
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=3,write_TS(Y)=1,write_TS(Z)=1
T2: read_item(Z)
TS(T2) > write_TS(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T2)) = 7
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=1,write_TS(Z)=1
T1: read_item(Y)
TS(T1) > write_TS(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=3,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=1,write_TS(Z)=1
T1: write_item(Y)
TS(T1) = read_TS(Y) and TS(T1) > write_TS(Y)
Execute write_item(Y)
write_TS(Y) <- max(read_TS(Y),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=3,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=3,write_TS(Z)=1
T2: read_item(Y)
TS(T2) > write_TS(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T2)) = 7
read_TS(X)=3,read_TS(Y)=7,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=3,write_TS(Z)=1
T2: write_item(Y)
TS(T2) = read_TS(Y) and TS(T2) > write_TS(Y)
Execute write_item(Y)
write_TS(Y) <- max(write_TS(Y),TS(T2)) = 7
read_TS(X)=3,read_TS(Y)=7,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=7,write_TS(Z)=1
T2: read_item(X)
TS(T2) > write_TS(X)
Execute read_item(X)
Set read_TS(X) <- max(read_TS(X),TS(T2)) = 7
read_TS(X)=7,read_TS(Y)=7,read_TS(Z)=7,write_TS(X)=3,write_TS(Y)=3,write_TS(Z)=1
T2: write_item(X)
TS(T2) = read_TS(X) and TS(T2) > write_TS(X)
Execute write_item(X)
write_TS(X) <- max(write_TS(X),TS(T2)) = 7
read_TS(X)=7,read_TS(Y)=7,read_TS(Z)=7,write_TS(X)=7,write_TS(Y)=7,write_TS(Z)=1

Result: Schedule F executes successfully.

20.26 Repeat Exercise 20.25, but use the multiversion timestamp ordering method.

Answer:

Let us assume the same timestamp values as in the solution for Exercise 18.22 above. To refer to versions, we use X, Y, Z to reference the original version (value) of each item, and then use indexes (1, 2, ...) to refer to newly written version (for example, X1, X2, ...).

(a) Applying multiversion timestamp ordering to Schedule E:
Initial values (new values are shown after each operation):
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
TS(T1)=6, TS(T2)=1, TS(T3)=4 (These do not change)
T2: read_item(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T2: read_item(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T2)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T2: write_item(Y)
TS(T2) = read_TS(Y)
Execute write_item(Y) (by creating a new version Y1 of Y)
write_TS(Y1) <- TS(T2) = 1,
read_TS(Y1) <- TS(T2) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Y1)=1,read_TS(Z)=1,
write_TS(X)=0,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T3: read_item(Y)
Execute read_item(Y) by reading the value of version Y1
read_TS(Y1) <- max(read_TS(Y1),TS(T3)) = 4
read_TS(X)=0,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Z)=1,
write_TS(X)=0,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T3: read_item(Z)
Execute read_item(Z)
read_TS(Z) <- max(read_TS(Z),TS(T3)) = 4
read_TS(X)=0,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Z)=4,
write_TS(X)=0,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T1: read_item(X)
Execute read_item(X)
read_TS(X) <- max(read_TS(X),TS(T1)) = 6
read_TS(X)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Z)=4,
write_TS(X)=0,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T1: write_item(X)
Execute write_item(X) (by creating a new version X1 of X)
write_TS(X) <- TS(T1) = 6,
read_TS(X) <- TS(T1) = 6
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Z)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T3: write_item(Y)
Execute write_item(Y) (by creating a new version Y2 of Y)
write_TS(Y2) <- TS(T3) = 4,
read_TS(Y2) <- TS(T3) = 4
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=4,
read_TS(Z)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Z)=0
T3: write_item(Z)
Execute write_item(Z) (by creating a new version Z1 of Z)
write_TS(Z1) <- TS(T3) = 4,
read_TS(Z1) <- TS(T3) = 4

read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=4,
read_TS(Z)=4,read_TS(Z1)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Z)=0,write_TS(Z1)=4
T2: read_item(X)
Execute read_item(X) by reading the value of the initial version X
read_TS(X) <- max(read_TS(X),TS(T3)) = 6
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=4,
read_TS(Z)=4,read_TS(Z1)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Z)=0,write_TS(Z1)=4
T1: read_item(Y)
Execute read_item(Y) by reading the value of version Y2
read_TS(Y2) <- max(read_TS(Y2),TS(T3)) = 6
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=6,
read_TS(Z)=4,read_TS(Z1)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Z)=0,write_TS(Z1)=4
T1: write_item(Y)
Execute write_item(Y) (by creating a new version Y3 of Y)
write_TS(Y3) <- TS(T3) = 4,
read_TS(Y2) <- TS(T3) = 4
read_TS(X)=6,read_TS(X1)=6,read_TS(Y)=1,read_TS(Y1)=4,read_TS(Y2)=6,
read_TS(Y3)=6,read_TS(Z)=4,read_TS(Z1)=4,
write_TS(X)=0,write_TS(X1)=6,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=4,
write_TS(Y2)=6,write_TS(Z)=0,write_TS(Z1)=4
T2: write_item(X)
Abort and Rollback T2 since read_TS(X) >TS(T2)

Result: Since T3 had read the value of Y that was written by T2, T3 should also be aborted and rolled by the recovery technique (because of cascading rollback); hence, all effects of T2 and T3 would also be erased and only T1 would finish execution.

(b) Applying timestamp ordering to Schedule F:
Initial values (new values are shown after each operation):
read_TS(X)=0,read_TS(Y)=0,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
TS(T1)=3, TS(T2)=7, TS(T3)=1 (These do not change)
T3: read_item(Y)
Execute read_item(Y)
Set read_TS(Y) <- max(read_TS(Y),TS(T3)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=0,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T3: read_item(Z)
Execute read_item(Z)
Set read_TS(Z) <- max(read_TS(Z),TS(T3)) = 1
read_TS(X)=0,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T1: read_item(X)
Execute read_item(X)
read_TS(X) <- max(read_TS(X),TS(T1)) = 3
read_TS(X)=3,read_TS(Y)=1,read_TS(Z)=1,write_TS(X)=0,write_TS(Y)=0,write_TS(Z)=0
T1: write_item(X)
Execute write_item(X) by creating a new version X1 of X
write_TS(X1) <- TS(T1) = 3, read_TS(X1) <- TS(T1) = 3
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=1,read_TS(Z)=1,

write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Z)=0
T3: write_item(Y)
Execute write_item(Y) by creating a new version Y1 of Y
write_TS(Y1) <- TS(T3) = 1, read_TS(Y1) <- TS(T3) = 1
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=1,read_TS(Y1)=1,read_TS(Z)=1,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0
T3: write_item(Z)
Execute write_item(Z) by creating a new version Z1 of Z
write_TS(Z1) <- TS(T3) = 1, read_TS(Z1) <- TS(T3) = 1
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=1,read_TS(Y1)=1,read_TS(Z)=1,
read_TS(Z1)=1,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0,
write_TS(Z1)=1
T2: read_item(Z)
Execute read_item(Z) by reading the value of version Z1
Set read_TS(Z1) <- max(read_TS(Z1),TS(T2)) = 7
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=1,read_TS(Y1)=1,read_TS(Z)=1,
read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0,
write_TS(Z1)=1
T1: read_item(Y)
Execute read_item(Y) by reading the value of version Y1
Set read_TS(Y1) <- max(read_TS(Y1),TS(T1)) = 3
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Z)=1,
read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Z)=0,
write_TS(Z1)=1
T1: write_item(Y)
Execute write_item(Y) by creating a new version Y2 of Y
write_TS(Y2) <- TS(T1) = 3, read_TS(Y2) <- TS(T1) = 3
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Y2)=3,
read_TS(Z)=1,read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=3,
write_TS(Z)=0,write_TS(Z1)=1
T2: read_item(Y)
Execute read_item(Y) by reading the value of version Y2
Set read_TS(Y2) <- max(read_TS(Y2),TS(T2)) = 7
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Y2)=7,
read_TS(Z)=1,read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=3,
write_TS(Z)=0,write_TS(Z1)=1
T2: write_item(Y)
Execute write_item(Y) by creating a new version Y3 of Y
write_TS(Y3) <- TS(T2) = 7, read_TS(Y3) <- TS(T2) = 7
read_TS(X)=3,read_TS(X1)=3,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Y2)=7,
read_TS(Y3)=7,read_TS(Z)=1,read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=3,
write_TS(Y3)=7,write_TS(Z)=0,write_TS(Z1)=1
T2: read_item(X)
Execute read_item(X) by reading the value of version X1
Set read_TS(X1) <- max(read_TS(X1),TS(T2)) = 7
read_TS(X)=3,read_TS(X1)=7,read_TS(Y)=3,read_TS(Y1)=1,read_TS(Y2)=7,
read_TS(Y3)=7,read_TS(Z)=1,read_TS(Z1)=7,

write_TS(X)=0,write_TS(X1)=3,write_TS(Y)=0,write_TS(Y1)=1,write_TS(Y2)=3,
write_TS(Y3)=7,write_TS(Z)=0,write_TS(Z1)=1
T2: write_item(X)
Execute write_item(X) by creating a new version X2 of X
write_TS(X2) <- TS(T2) = 7, read_TS(X2) <- TS(T2) = 7
read_TS(X)=3,read_TS(X1)=7,read_TS(X2)=7,read_TS(Y)=3,read_TS(Y1)=1,
read_TS(Y2)=7,read_TS(Y3)=7,read_TS(Z)=1,read_TS(Z1)=7,
write_TS(X)=0,write_TS(X1)=3,write_TS(X2)=7,write_TS(Y)=0,write_TS(Y1)=1,
write_TS(Y2)=3,write_TS(Y3)=7,write_TS(Z)=0,write_TS(Z1)=1

Result: Schedule F executes successfully.

18.27 Solution pending.

18.28 Solution pending.

18.29 Solution pending.

CHAPTER 19: DATABASE RECOVERY TECHNIQUES

Answers to Selected Exercises

19.21 Suppose that the system crashes before the [read_item,T3,A] entry is written to the log in Figure 19.1(b); will that make any difference in the recovery process?

Answer:

There will be no difference in the recovery process, because read_item operations are needed only for determining if cascading rollback of additional transactions is necessary.

19.22 Suppose that the system crashes before the [write_item,T2,D,25,26] entry is written to the log in Figure 19.1(b); will that make any difference in the recovery process?

Answer:

Since both transactions T2 and T3 are not yet committed, they have to be rolled back during the recovery process.

19.23 Figure 19.7 shows the log corresponding to a particular schedule at the point of a system crash for the four transactions T1, T2, T3, and T4 of Figure 19.4. Suppose that we use the immediate update protocol with checkpointing. Describe the recovery process from the system crash. Specify which transactions are rolled back, which operations in the log are redone and which (if any) are undone, and whether any cascading rollback takes place.

Answer:

First, we note that this schedule is not recoverable, since transaction T4 has read the value of B written by T2, and then T4 committed before T2 committed. Similarly, transaction T4 has read the value of A written by T3, and then T4 committed before T3 committed. The [commit, T4] should not be allowed in the schedule if a recoverable protocol is used, but should be postponed till after T2 and T3 commit. For this problem, let us assume that we can roll back a committed transaction in a non-recoverable schedule, such as the one shown in Figure 21.7.
By using the procedure RIU_M (recovery using immediate updates for a multiuser environment), the following result is obtained:
From Step 1 of procedure RIU_M, T2 and T3 are the active transactions. T1 was committed before the checkpoint and hence is not involved in the recovery.
From Step 2, the operations that are to be undone are:
[write_item,T2,D,25]
[write_item,T3,A,30]
[write_item,T2,B,12]
Note that the operations should be undone in the reverse of the order in which they were written into the log. Now since T4 read item B that as written by T2 and read item A that as written by T3, and since T2 and T3 will be rolled back, by the cascading rollback rule, T4 must be also rolled back. Hence, the following T4 operations must also be undone:
[write_item,T4,A,20]
[write_item,T4,B,15]
(Note that if the schedule was recoverable and T4 was committed, then from Step 3, the operations that are to be redone would have been:
[write_item,T4,B,15]

[write_item,T4,A,20]
In our case of non-recoverable schedule, no operations need to be redone in this example.)
At the point of system crash, transactions T2 and T3 are not committed yet. Hence, when T2 is rolled back, transaction T4 should also be rolled back as T4 reads the values of items B and A that were written by transactions T2 and T3. The write operations of T4 have to be undone in their correct order. Hence, the operations are undone in the following order:
[write_item,T2,D,25]
[write_item,T4,A,20]
[write_item,T3,A,30]
[write_item,T4,B,15]
[write_item,T2,B,12]

19.24 Suppose that we use the deferred update protocol for the example in Figure 19.7. Show how the log would be different in the case of deferred update by removing the unnecessary log entries; then describe the recovery process, using your modified log. Assume that only redo operations are applied, and specify which operations in the log are redone and which are ignored.

Answer:

In the case of deferred update, the write operations of uncommitted transactions are not recorded in the database until the transactions commit. Hence, the write operations of T2 and T3 would not have been applied to the database and so T4 would have read the previous (committed) values of items A and B, thus leading to a recoverable schedule.
By using the procedure RDU_M (deferred update with concurrent execution in a multiuser environment), the following result is obtained:
The list of committed transactions T since the last checkpoint contains only transaction T4. The list of active transactions T' contains transactions T2 and T3.
Only the WRITE operations of the committed transactions are to be redone. Hence, REDO is applied to:
[write_item,T4,B,15]
[write_item,T4,A,20]
The transactions that are active and did not commit i.e., transactions T2 and T3 are canceled and must be resubmitted. Their operations do not have to be undone since they were never applied to the database.

19.25 How does checkpointing in ARIES differ from checkpointing as described in Section 19.1.4?

Answer:

The main difference is that with ARIES, main memory buffers that have been modified are not flushed to disk. ARIES, however writes additional information to the LOG in the form of a Transaction Table and a Dirty Page Table when a checkpoint occurs.

19.26 How are log sequence numbers used by ARIES to reduce the amount of REDO work needed for recovery? Illustrate with an example using the information shown in Figure 19.6. You can make your own assumptions as to when a page is written to disk.

Answer:

Since ARIES writes the Dirty Page Table to the LOG at checkpoint time, ARIES can use the LSN (log sequence number) information stored in that table and the data pages during recovery.
REDO only has to start after the point where all prior changes are known to be in the database. Hence, REDO can start at the place in the LOG that corresponds to the smallest LSN from the Dirty Page Table.
In Figure 21.6, since the smallest LSN in the Dirty Page Table is 1, REDO must start at location 1 in the LOG. Let's assume that when Transaction T2 performed the update of page C, page C was written to disk. So, page C on disk would have an associated LSN of 7. When REDO starts at location 1 in the LOG (LSN of 1), page C would be read into the main memory buffer, however no change is necessary since the the LSN of page C is 7, which is larger than LSN of 1 for the current LOG update operation. Proceeding with location 2 in the LOG, page B would be brought into memory. If the LSN of page B is less than 2, then page B would be updated with the corresponding change in location 2 in the LOG. Similarly for location 6, page A would be updated. For location 7, Page C need not be updated since the LSN of page C (i.e., 7) is not less than the LSN of the current LOG update operation.
21.27 What implications would a no-steal/force buffer management policy have on checkpointing and recovery?

No-steal means that the cache (buffer) page updated by a transaction cannot be written to disk before the transaction commits. Force means that updated pages are written to disk when a transaction commits.

With No-steal, the checkpoint scheme that writes all modified main memory buffers to disk would not be able to write pages updated by uncommitted transactions.

With Force, once a transaction is done, its updates would be pushed to disk. If there is a failure during this, REDO is still needed, however, no UNDO is needed since uncommitted updates are never propagated to disk.

19.28 (b)
19.29 (b)
19.30 (b)
19.31 (a)
19.32 c
19.33 (a)
19.34 c
19.35 (d)
19.36 (b)
19.37 (b)

CHAPTER 20: CONCEPTS FOR OBJECT DATABASES

Answers to Selected Exercises

20.13 Convert the example of GEOMETRY_OBJECTS given in section 20.4.1from the functional notation to the notation given in Figure 20.3 that distinguishes between attributes and operations. Use the keyword INHERIT to show that one class inherits from another class.

Answer:

(GEOMETRY_OBJECTS to attribute/operation)

```
define class GEO_OBJECT:
type tuple ( shape : enumerated(rectangle, triangle, circle);
refpoint : tuple ( x: float,
y: float) );
operations area : float;
end GEO_OBJECT;
define class RECTANGLE:
INHERITS GEO_OBJECT;
type tuple ( width : float,
height: float);
operations square? : boolean;
end RECTANGLE;
define class TRIANGLE:
INHERITS GEO_OBJECT;
type tuple ( side1 : float,
side2 : float,
angle : float );
end TRIANGLE;
define class CIRCLE:
INHERITS GEO_OBJECT;
type tuple ( radius : float );
operations diameter : float;
ci rcumference : float;
end CIRCLE;
```

20.14 Compare inheritance in the EER model (see Chapter 4) to inheritance in the OO model described in Section 20.4

Answer:

(EER inheritance vs. OO inheritance)
EER inheritance has closer correspondence to the real world, in the sense that it is synthesized from facts about tangible entities, rather than imposed through software engineering process. At the EER level, the possible ambiguities of multiple inheritance may be deferred until implementation.
Object-oriented database design emphasizes abstract structures and methods; objects need not exist. Polymorphism and visibility properties drive the design. OO inheritance allows membership in many different sets and extents. In fact, membership is distinguished from properties, as seen in type and extent inheritance.

20.15 Consider the UNIVERSITY EER schema of Figure 4.10. Think of what operations are needed for the entity types/classes in the schema. Do not consider constructor and destructor operations.

Answer:

(University database operations)
define class PERSON:

operations change-address (new : Address) : Address;
define class COURSE:

operations new-section (Yr, Qtr : integer) : Section;
get-current : set (cur-sec : Current_Section);
change-dept (new-dept : Department) : Course;
define class STUDENT:
operations change-major (new : Major) : Student;
register (sec : Current_Section) : boolean;
g et-transcript : set (tuple (course-taken : Course;
grade : char;));
define class FACULTY:

operations call-comm-meeting (S : Student) : set (Faculty);
compute-support : float;
change-dept (new-dept : Department) : Faculty;
make-chairman (dept : Department) : Faculty;
Many other operations are possible, as well.

20.16 Consider the COMPANY ER schema of Figure 3.2. Think of what operations are needed for the entity types/classes in the schema. Do not consider constructor and destructor operations.

Answer:

(Company database operations)
define class EMPLOYEE:

operations raise-salary (pct : float) : boolean;
change address (new : Address) : boolean;
get-age : integer;
get-supervisor : Employee;
get-supervisees : set (Employee);
show-projects : set (tuple (Pname : Project,
hours : integer));
show-dependents : set (Dependent);
define class DEPARTMENT:

operations count-emps : integer;
get-manager : Employee;
get-locations : set (String);
list-controlled-projects : set (Project);
define class PROJECT:

```
operations get-department : Department;
show-emp-hours : float;
show-dept-contributions : set ( tuple
( D : Department,
set ( tuple ( emp : Employee,
hrs : integer )
) ) );
```

Many other operations are possible, as well.

CHAPTER 21: OBJECT DATABASE STANDARDS, LANGUAGES, AND DESIGN

Answers to Selected Exercises

21.15 Map the COMPANY ER schema of Figure 3.2 into ODL classes. Include appropriate methods for each class.

Answer:

```
class Employee
( extent employees
key ssn
{
attribute struct name {string fname, string mname, string lname}
name;
attribute string ssn;
attribute date bdate;
attribute enum Gender{M, F} sex;
attribute string address;
attribute float salary;
attributte 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_valid);
};
class Project
( extent projects
key pname, pnumber
{
attribute string pname;
attribute short pnumber;
attributte 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
{
attributte 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;
attributte Employee suporter;
short age();
};
```

12.18 Queries from exercises 7.18 (and 8.13) specified in OQL.

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

```
select e.name.fname
from e in employee
where e.works_for .dnumber = 5 and
( 'Product X' in
(select h.work_on.pname
from h in e.work
where h.hours > 10));
```

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

```
select k.suporter.fname
from k in dependents
where k.name = k.suporter.fname
```

(c) List the name of employees who are all directly supervised by 'Franklin Wong'.

```
select e.fname
from e in employee
where e.supervisor.name.fname = 'Franklin' and e.supervisor.name.lname = 'Wong';
```

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

```
select projname, tot_hours : sum (select p.h.hours from p in partition)
from h in hours_worked
group by projname: w.work_on.pname;
```

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

select e.name.fname
from e in employee
where for all g in (select p.pnumber
from p in projects)
: g.pnumber in (select e.work.work_on.pnumber);

(f) Retrieve the names of all employees who do not work on any project.
select e.name.fname
from e in employee
where for all g in (select p.pnumber
from p in projects)
: g.pnumber not in (select e.work.work_on.pnumber);

(g) For each department, retrieve the department name, and the average salary of employees
working in that department.
select deptname, avgsal : avg (select p.e.salary from p in partition)
from e in employee
group by deptname: e.works_for.dname;

(h) Retrieve the average salary of all female employees.
avg (select e.salarey
from e in employee
where e.sex = 'M');

CHAPTER 22: OBJECT-RELATIONAL AND EXTENDED-RELATIONAL SYSTEMS

No Exercises.

CHAPTER 23: DATABASE SECURITY AND AUTHORIZATION

Answers to Selected Exercises

23.22 Consider the relational database schema of Figure 5.5. Suppose that all the relations were created by (and hence are owned by) user X, who wants to grant the following privileges to user accounts A, B, C, D, and E:

(a) Account A can retrieve or modify any relation except DEPENDENT and can grant any of these privileges to other users.

(b) Account B can retrieve all the attributes of EMPLOYEE and DEPARTMENT except for SALARY, MGRSSN, and MGRSTARTDATE.

(c) Account C can retrieve or modify WORKS_ON but can only retrieve the FNAME, MINIT, LNAME, SSN attributes of EMPLOYEE and the PNAME, PNUMBER attributes of PROJECT.

(d) Account D can retrieve any attribute of EMPLOYEE or DEPENDENT and can modify DEPENDENT.

(e) Account E can retrieve any attribute of EMPLOYEE but only for EMPLOYEE tuples that have DNO = 3.

Write SQL statements to grant these privileges. Use views were appropriate.

Answer:

(a)
```
GRANT SELECT, UPDATE
ON EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON
TO USER_A
WITH GRANT OPTION;
```

(b)
```
CREATE VIEW EMPS AS
SELECT FNAME, MINIT, LNAME, SSN, BDATE, ADDRESS, SEX,
SUPERSSN, DNO
FROM EMPLOYEE;
GRANT SELECT ON EMPS
TO USER_B;
CREATE VIEW DEPTS AS
SELECT DNAME, DNUMBER
FROM DEPARTMENT;
GRANT SELECT ON DEPTS
TO USER_B;
```

(c)
```
GRANT SELECT, UPDATE ON WORKS_ON TO USER_C;
CREATE VIEW EMP1 AS
SELECT FNAME, MINIT, LNAME, SSN
FROM EMPLOYEE;
GRANT SELECT ON EMP1
```

```
TO USER_C;
CREATE VIEW PROJ1 AS
SELECT PNAME, PNUMBER
FROM PROJECT;
GRANT SELECT ON PROJ1
TO USER_C:
```

(d)

```
GRANT SELECT ON EMPLOYEE, DEPENDENT TO USER_D;
GRANT UPDATE ON DEPENDENT TO USER_D;
```

(e)

```
CREATE VIEW DNO3_EMPLOYEES AS
SELECT * FROM EMPLOYEE
WHERE DNO=3;
GRANT SELECT ON DNO3_EMPLOYEES TO USER_E;
```

23.23 Suppose that privilege (a) of exercise 23.1 is to be given with GRANT OPTION but only so that account A can grant it to at most five accounts, and each of these accounts can propagate the privilege to other accounts but without the GRANT OPTION privilege. What would the horizontal and vertical propagation limits be in this case?

Answer:

The horizontal propagation limit granted to USER_A is 5. The vertical propagation limit granted to USER_A is level 1, so that USER_A can then grant it with level 0 vertical limit (i.e. without the GRANT OPTION) to at most five users, who then cannot further grant the privilege.

23.24 Consider the relation shown in Figure 23.2(d). How would it appear to a user with classification U? Suppose a classification U user tries to update the salary of "Smith" to $50,000; what would be the result of this action?

Answer:

EMPLOYEE would appear to users within classification U as follows:
NAME SALARY JobPerformance TC
Smith U null U null U U
If a classification U user tried to update the salary of Smith to $50,000, a third polyinstantiation of Smith's tuple would result as follows:
NAME SALARY JobPerformance TC
Smith U 40000 C Fair S S
Smith U 40000 C Excellent C C
Smith U 50000 U null U U
Brown C 80000 S Good C S

CHAPTER 24: ENHANCED DATA MODELS FOR ADVANCED APPLICATIONS

Answers to Selected Exercises

24.15 Consider the COMPANY database described in Figure 5.6. Using the syntax of Oracle triggers, write active rules to do the following:

(a) Whenever an employee's project assignments are changed, check if the total hours per week spent on the employee's projects are less than 30 or greater than 40; if so, notify the employee's direct supervisor.

(b) Whenever an EMPLOYEE is deleted, delete the PROJECT tuples and DEPENDENT tuples related to that employee, and if the employee is managing a department or supervising any employees, set the MGRSSN for that department to null and set the SUPERSSN for those employees to null.

Answers:
(a) We assume that a procedure TELL_SUPERVISOR(ARGSSN) has been created. This procedure looks for an employee whose SSN matches the procedure's AGRSSN argument and it notifies the supervisor of that employee.

```
CREATE TRIGGER INFORM_SUPERVISOR_ABOUT_HOURS
AFTER UPDATE OF HOURS ON WORKS_ON
FOR EACH ROW
WHEN ((SELECT SUM(HOURS)
FROM WORKS_ON
WHERE ESSN = NEW.ESSN) < 30
OR
(SELECT SUM(HOURS)
FROM WORKS_ON
WHERE ESSN = NEW.ESSN) > 40)
TELL_SUPERVISOR (NEW.ESSN);
```

(b)
```
CREATE TRIGGER DELETE_IN_CASCADE
AFTER DELETE ON EMPLOYEE
FOR EACH ROW
BEGIN
DELETE FROM WORKS_ON
WHERE ESSN = OLD.SSN;
DELETE FROM DEPENDENT
WHERE ESSN = OLD.SSN;
UPDATE EMPLOYE
SET SUPERSSN = 'null'
WHERE SUPERSSN = OLD.SSN;
END;
```

24.16 Repeat 24.15 but use the syntax of STARBURST active rules.

Answer:

(a) We assume that a procedure TELL_SUPERVISOR(ARGSSNS) has been created. This procedure looks for employees whose SSN matches the social security numbers passed by the procedure's AGRSSNS argument and it notifies supervisors of those employees.

```
CREATE RULE INFORM_SUPERVISOR_ABOUT_HOURS ON WORKS_ON
WHEN UPDATED (HOURS)
THEN TELL_SUPERVISOR (SELECT DISTINCT ESSN FROM
WORKS_ON AS W WHERE
((SELECT SUM(HOURS) FROM WORKS_ON AS R
WHERE W.ESSN = R.ESSN) < 30)
OR
(SELECT SUM(HOURS) FROM WORKS_ON AS R
WHERE W.ESSN = R.ESSN) > 40))
AND
W.ESSN IN (SELECT ESSN FROM NEWUPDATED);
```

(b)

```
CREATE RULE DELETE_IN_CASCADE ON EMPLOYEE
WHEN DELETED
THEN DELETE FROM WORKS_ON AS W
WHERE W.ESSN IN (SELECT ESSN FROM DELETED AS D
WHERE D.ESSN = W.ESSN);
DELETE FROM DEPENDENT AS P
WHERE P.ESSN IN (SELECT ESSN FROM DELETED AS D
WHERE D.ESSN = P.ESSN);
UPDATE EMPLOYEE AS E
SET E.SUPERSSN = ‘null’;
WHERE E.SSN IN (SELECT ESSN FROM DELETED AS D
WHERE D.ESSN = E.ESSN);
```

24.17 Consider the relational schema shown in Figure 24.18. Write active rules for keeping the SUM_COMMISSIONS attribute of SALES_PERSON equal to the sum of the COMMISSION attribute in SALES for each sales person. Your rules should also check if the SUM_COMMISSIONS exceeds 100000; if it does, call a procedure NOTIFY_MANAGER(S_ID). Write both statement-level rules in STARBURST notation and row-level rules in Oracle.

Answer:

Oracle notation rules:

```
CREATE TRIGGER KEEP_EM_SAME
AFTER INSERT OR UPDATE OF COMMISION ON SALES
FOR EACH ROW
UPDATE SALES_PERSON
SET SUM_COMISSIONS = (SELECT SUM (COMISSION)
FROM SALES
WHERE S_ID = NEW.S_ID);
CREATE TRIGGER NOTIFY_MANAGEMENT
AFTER INSERT OR UPDATE OF SUM_COMISSIONS OF SALES_PERSON
FOR EACH ROW
WHEN ((SELECT SUM(COMISSION)
FROM SALES_PERSON
WHERE SALES_PERSON_ID = NEW.SALES_PERSON_ID) >
100000
NOTIFY_MANAGER(NEW.SALES_PERSON_ID);
```

Starburst notation rules:

```
CREATE RULE KEEP_EM_SAME ON SALES
```

```
WHEN INSERTED OR UPDATED(COMISSION)
THEN UPDATE SALES_PERSON AS S
SET S.SUM_COMISSIONS = ( SELECT SUM(COMISSIONS)
FROM SALES AS L
WHERE S.S_ID = L.S_ID)
WHERE S.SALESPERSON_ID IN ((SELECT S_ID FROM UPDATED)
OR (SELECT S_ID FROM INSERTED));
CREATE RULE NOTIFY_MANAGEMENT ON SALES_PERSON
WHEN INSERTED OR UPDATED(SUM_COMISSIONS)
THEN NOTIFY_MANAGER(SELECT SALESPERSON_ID
FROM SALES_PERSON AS S
WHERE S.SUM_COMISSIONS > 100000)
AND
S.SALESPERSON_ID IN
((SELECT SALESPERSON_ID FROM
UPDATED)
OR (SELECT SALESPERSON_ID FROM
INSERTED)));
```

We assumed that there would be no deletions of sales. If deletion of sales were a possibility, we would simply enhance the above rules to accommodate the delete option as well.

24.18 Consider the UNIVERSITY EER schema of Figure 4.10. Write some rules (in English) that could be implemented via active rules to enforce some common integrity constraints that you think are relevant to this application.

Answer:

RULE 1: Every time a graduate student receives a grade of C of lower, notify his or her advisor.
RULE 2: When a faculty member becomes a department chair, raise her or his salary 15%.
RULE 3: If a student changes his or her major, delete all of his or her minors, and notify the student to sign up for minors again.

24.19 Discuss which of the updates that created each of the tuples shown in Figure 24.9 were applied retroactively and which were applied proactively.

Answer:

- V1, V6, V8, V9, and V11 are all products of proactive updates, because their TST occurred before their
VST.
- V2, V3, V5, V7, and V10 are all products of retroactive updates, because their TST occurred after their
VST.

CHAPTER 27: DATA MINING CONCEPTS

Answers to Selected Exercises

27.14 Apply the Apriori algorithm to the following data set:

Trans ID	Items Purchased
101	milk, bread, eggs
102	milk, juice
103	juice, butter
104	milk, bread, eggs
105	coffee, eggs
106	coffee
107	coffee, juice
108	milk, bread, cookies, eggs
109	cookies, butter
110	milk, bread

The set of items is {milk, bread, cookies, eggs, butter, coffee, juice}. Use 0.2 for the minimum support value.

Answer:

First, we compute the support for 1-item sets
(e.g., milk appears in 5 out of the 10 transactions, support is 0.5):

1-ITEM SETS	SUPPORT
milk	0.5
bread	0.4
eggs	0.4
coffee	0.3
juice	0.3
cookies	0.2
butter	0.2

The min support required is 0.2, so all 1-item sets satisfy this requirement, i.e. they are all frequent.

For the next iteration, we examine 2-item sets composed of the frequent 1-item sets. The number of potential 2-item sets is 21 (i.e., 7 items taken 2 at a time). The 2-item sets that satisfy the min support of 0.2 are the following:

2-ITEM SETS	SUPPORT
milk,bread	0.4
milk,eggs	0.3
bread,eggs	0.3

For the next iteration, we examine 3-item sets composed of the frequent 2-item sets. The 3-item sets that satisfy the min support of 0.2 are the following:

3-ITEM SETS SUPPORT

milk,bread,eggs 0.3

27.15 Show two rules that have a confidence of 0.7 or greater for an itemset containing three items from Exercise 14.

Answer:

There is only one frequent itemset of size 3, i.e., {milk,bread,eggs}. We can try the rule milk,eggs -> bread. The confidence of this rule is 0.3/0.3 which exceeds the min confidence value of 0.7. Another rule we can try is bread -> milk,eggs. The confidence of this rule is 0.3/0.4 which again satisfies the min confidence requirement.

27.16 For the Partition algorithm, prove that any frequent itemset in the database must appear as a local frequent itemset in at least one partition.

Answer:

We can do a proof by contradiction.
    Assume M transactions,
        N partitions, wlog each contains M/N transactions
        frequent itemset I with support S,
         where S * M = number of transactions containing I,

    We know that since I is a frequent itemset, then S >= min_support
    or equivalently, S * M >= min_support * M.

    Now assume that I is not frequent within any of the N partitions, Pi,
    i.e., the support within a partition Pi is Si which is < min_support, or
    equivalently Si * M/N < min_support * M/N.
    Hence,

    (S1 * M/N) + (S2 *M/N) + ... + (SN * M/N) < N * (min_support * M/N)

    (S1 * M/N) + (S2 *M/N) + ... + (SN * M/N) < min_support * M

    This contradicts the fact that the support of itemset I should be
    >= min_support or equivalently that the number of transactions containing
    I be >= min_support * M.

27.17 Show the FP tree that would be made for the data from Exercise 14.

Answer:

The FP tree is the following:

```
        ITEM    SUPPORT
 ........milk       5
 . ......bread      4
 . . ....eggs       4
 . . .   coffee     3  ....................................
 . . .   juice      3 ......                              .
```

```
. . . ..cookies    2      .                          .
. . . . butter     2 .............                 .
. . . .                 .      .                     .
. . . .                 .      .                     .
. . . .                 .      .      --------        .
. . . .                 .      .     |  NULL  |        .
. . . .         ----------.------.----- -------- ---------.----------
. . . .        /          .      .     /             \  .         \
. . . .       V            .      .    V               V V          V
. . . .   -------          .      . --------            --------     ---------
... . ..>|milk :5|        . .....>|juice :1|..        |coffee:3|   |cookies:1|<...
  . . .   ------- -----   . .    . --------  .      --- --------     ---------    .
  . . .      |        \  . .    .   |      ...../.......   |          |        .
  . . .      V         V V .    V   V          V       V   V          V        .
  . . .   -------     -------     --------     ------     -------     --------    .
  ......>|bread:4|   |juice:1|   |butter:1|   |eggs:1|   |juice:1|   |butter:1|   .
    . .   -------     -------     --------     ------     -------     --------    .
    . .     |                      .         ^                     ^       .
    . .     V                      ....................................       .
    . .   -------                              .                           .
    ....>|eggs :3| ...............................                           .
      .   -------                                                          .
      .     |                                                             .
      .     V                                                              .
      .   ---------                                                        .
      ..>|cookies:1| .............................................................
          ---------
```

27.18 Apply the FP-growth algorithm to the FP tree from Exercise 17 and show the frequent itemsets.

Answer:

We start with butter. There are two prefixes (juice:1) and (cookies:1).
    Since support is only 1, the conditional FP tree is empty.

    Next we consider cookies, the conditional pattern base is
    (milk, bread, eggs:1). Once again, the support is too low.

    Next we consider juice. The two prefixes for juice are (milk:1) and (coffee:1)
    but since the support is only 1, the conditional FP tree is empty.

    Next consider coffee but the conditional pattern base is empty.

    Next consider eggs. We have (milk, bread :3) and (coffee :1). The support
    for coffee is too low so it is eliminated but not for milk and bread.
    The algorithm is called recursively on the FP tree with milk and bread nodes,
    using a beta value of eggs. So, the patterns returned are (milk, eggs: 3),
    (bread, eggs :3) and (milk, bread, eggs :3).

    Next consider bread. The prefix is (milk :4) and we produce the pattern
    (milk, bread :4)

Finally, consider milk, which has no prefix, so we are done.

If we put together all the patterns found, we end up with
(milk:5), (bread:4),(eggs:4),(coffee:3),(cookies:2),(butter:2),
(juice:2), (milk, eggs: 3), (bread, eggs :3), (milk, bread, eggs :3)
and (milk, bread :4).

27.19 Apply the classification algorithm to the following set of data records. The class attribute is Repeat Customer.

RID	**Age**	**City**	**Gender**	**Education**	**Repeat Customer**
101	20..30	NY	F	College	YES
102	20..30	SF	M	Graduate	YES
103	31..40	NY	F	College	YES
104	51..30	NY	F	College	NO
105	31..40	LA	M	High school	NO
106	41..50	NYY	F	College	YES
107	41..50	NY	F	Graduate	YES
108	20..30	LA	M	College	YES
109	20..30	NY	F	High school	NO
110	20..30	NY	F	college	YES

Answer:

We start by computing the entropy for the entire set. We have 7 positive samples and 3 negative samples.
The entropy, I(7,3), is -(7/10 * log (7/10) + 3/10 * log(3/10)) = 0.88

We consider the first attribute AGE. There are 4 values for age
20..30 appears 5 times
I(s11, s21) = -(4/5 * log(4/5) + 1/5 * log(1/5)) = 0.72
31..40 appears 2 times
I(s12, s22) = -(1/2 * log(1/2) + 1/2 * log(1/2)) = 1
41..50 appears 2 times
I(s13, s23) = -(2/2 * log(2/2) = 0
51..60 appears 1 time
I(s14, s24) = -(1/1 * log(1/1) = 0

E(AGE) = 5/10 * 0.72 + 2/10 * 1 + 2/10 * 0 + 1/10 * 0 = 0.56
GAIN(AGE) = 0.88 - 0.56 = 0.32

We consider the second attribute CITY. There are 3 values for city
LA occurs 2 times
I(s11, s21) = -(1/2 * log(1/2) + 1/2 * log(1/2)) = 1
NY occurs 7 times
I(s12, s22) = -(2/7 * log(2/7) + 5/7 * log(5/7)) = 0.86
SF occurs 1 times
I(s13, s23) = -(1/1 * log(1/1) = 0

E(CITY) = 2/10 * 1 + 7/10 * 0.86 + 1/10 * 0 = 0.80
GAIN(CITY) = 0.88 - 0.80 = 0.08

We consider the third attribute GENDER. There are 2 values
F occurs 7 times
I(s11, s21) = -(2/7 * log(2/7) + 5/7 * log(5/7)) = 0.86
M occurs 3 times
I(s12, s22) = -(1/3 * log(1/3) + 2/3 * log(2/3)) = 0.92

E(GENDER) = 0.88
GAIN(GENDER) = 0

We consider the fourth attribute EDUCATION. There are 3 values
HS occurs 2 times
I(s11, s21) = -(2/2 * log(2/2) = 0
COLLEGE occurs 6 times
I(s12, s22) = -(1/6 * log(1/6) + 5/6 * log(5/6)) = 0.65
GRAD occurs 2 times
I(s13, s23) = -(2/2 * log(2/2) = 0

E(EDUCATION) = 0.39
GAIN(EDUCATION) = 0.49

The greatest gain is for the EDUCATION attribute.
The tree at this point would look like the following:

```
                 -------------------
                |   EDUCATION     |
                 -------------------
               /         |       \
        HS    /   COLLEGE |        \  GRAD
             /           |         \

RIDS:      {105,109}       {101,103,104,   {102,107}
       same class: NO      106,108,110}    same class: YES
```

Only the middle node is not a LEAF node, so continue with
those records and consider only the remaining attributes.
The entropy, I(5,1), is -(5/6* log (5/6) + 1/6 * log(1/6)) = 0.65

We consider the first attribute AGE. There are 4 values for age
20..30 appears 3 times
I(s11, s21) = -(3/3 * log(3/3) = 0
31..40 appears 1 time
I(s12, s22) = -(1/1 * log(1/1) = 0
41..50 appears 1 time
I(s13, s23) = -(1/1 * log(1/1) = 0
51..60 appears 1 time
I(s14, s24) = -(1/1 * log(1/1) = 0

E(AGE) = 0
GAIN(AGE) = 0.65

We consider the second attribute CITY. There are 2 values for city

NY occurs 1 time
I(s11, s21) = -(1/1 * log(1/1) = 0
SF occurs 5 times
I(s12, s22) = -(1/5 * log(1/5) + 4/5 * log(4/5)) = 0.72

E(CITY) = 0.60
GAIN(CITY) = 0.05

We consider the third attribute GENDER. There are 2 values
F occurs 5 times
I(s11, s21) = -(1/5 * log(1/5) + 4/5 * log(4/5)) = 0.72
M occurs 1 time
I(s12, s22) = -(1/1 * log(1/1) = 0

E(GENDER) = 0.60
GAIN(GENDER) = 0.05

The greatest gain is for the AGE attribute.
The tree at this point would look like the following
and we are finished.

```
                  ----------------------
                  |    EDUCATION       |
                  ----------------------
                 /         |           \
          HS    /   COLLEGE |            \  GRAD
               /           |             \
                        ----------------
RIDS:      {105,109}    |      AGE       |    {102,107}
       same class: NO   ----------------     same class: YES
                    /     /      |     \
                   /     /       |      \
          20..30 /     /31..40  |41..50 \  51..60
      {101,108,110}    {103}    {106}     {104}
     same class: YES     YES       YES       NO
```

27.20 Consider the following set of two-dimensional records:

RID	Dimension 1	Dimension 2
1	8	4
2	5	4
3	2	4
4	2	6
5	2	8
6	8	6

Also consider two different clustering schemes: (1) where Cluster 1 contains records {1, 2, 3} and Cluster 2 contains records {4, 5, 6} and (2) where Cluster 1 contains records {1, 6} and Cluster 2 contains records {2, 3, 4, 5}. Which scheme is better and why?

Answer:

Compare the error of the two clustering schemes. The scheme with the smallest error is better.

For SCHEME (1) we have C1 = {1,2,3} and C2 = {4,5,6}
M1 = ((8+5+2)/3, (4+4+4)/3) = (5,4)

$C1\_error = (8-5)^2 + (4-4)^2 + (5-5)^2 (4-4)^2 + (2-5)^2 + (4-4)^2$
$= 18$

For C2 we have
M2 = ((2+2+8)/3, (6+8+6)/3) = (4,6.66)

$C2\_error = (2-4)^2 + (6-6.66)^2 + (2-4)^2 (8-6.66)^2 + (8-4)^2 + (6-6.66)^2$
$= 26.67$

C1_error + C2_error = 44.67

For SCHEME (2) we have C1 = {1,6} and C2 = {2,3,4,5}
M1 = ((8+8)/2, (4+6)/2) = (8,5)

$C1\_error = (8-8)^2 + (4-5)^2 + (8-8)^2 (6-5)^2$
$= 2$

For C2 we have
M2 = ((5+2+2+2)/4, (4+4+6+8)/4) = (2.75,5.5)

C2_error =
$(5-2.75)^2 +(4-5.5)^2 +(2-2.75)^2 +(4-5.5)^2 +(2-2.75)^2 +(6-5.5)^2 +(2-2.75)^2 +(8-5.5)^2$

$= 17.74$

C1_error + C2_error = 19.74

SCHEME 2 is better since the error associated with it is less than that of SCHEME (1).

27.21 Use the K-means algorithm to cluster the data from Exercise 20. We can use a value of 3 for K and can assume that the records with RIDs 1, 3, and 5 are used for the initial cluster centroids (means).

Answer:

We start by specifying the centroid for each of the 3 clusters.
C1's centroid is (8,4) , i.e., record with rid = 1
C2's centroid is (2,4) , i.e., record with rid = 3

C3's centroid is (2,8) , i.e., record with rid = 5

We now place the remaining records in the cluster whose centroid is closest.

The distance between record 2, i.e., point (5,4), and centroid for C1 is

$\text{SQROOT}(|8-5|^2 + |4-4|^2) = 3$

The distance between record 2 and centroid for C2 is

$\text{SQROOT}(|2-5|^2 + |4-4|^2) = 3$

The distance between record 2 and centroid for C3 is

$\text{SQROOT}(|2-5|^2 + |8-4|^2) = 5$

Record 2 can be placed in either C1 or C2 since the distance from their respective centroids are the same. Let's choose to place the record in C1.

The distance between record 4, i.e., point (2,6), and centroid for C1 is

$\text{SQROOT}(|8-2|^2 + |4-6|^2) = 6.32$

The distance between record 4 and centroid for C2 is

$\text{SQROOT}(|2-2|^2 + |4-6|^2) = 2$

The distance between record 4 and centroid for C3 is

$\text{SQROOT}(|2-2|^2 + |8-6|^2) = 2$

Record 4 can be placed in either C2 or C3 since the distance from their respective centroids are the same. Let's choose to place the record in C2.

The distance between record 6, i.e., point (8,6), and centroid for C1 is

$\text{SQROOT}(|8-8|^2 + |4-6|^2) = 2$

The distance between record 6 and centroid for C2 is

$\text{SQROOT}(|2-8|^2 + |4-6|^2) = 6.32$

The distance between record 6 and centroid for C3 is

$\text{SQROOT}(|2-8|^2 + |8-6|^2) = 6.32$

Record 6 is closest to centroid of cluster C1 and is placed there.

We now recalculate the cluster centroids:

C1 contains records {1,2,6} with a centroid of
((8+5+8)/3, (4+4+6)/3) = (7, 4.67)

C2 contains records {3,4} with a centroid of
((2+2)/2, (4+6)/2) = (2, 5)

C3 contains record {5} with a centroid of (2, 8)

We now make a second iteration over the records, comparing the distance of each record with the new centroids and possibly moving records to new clusters. As it turns out, all records stay in their prior cluster assignment. Since there was no change, the algorithm terminates.

27.22 The K-means algorithm uses a similarly metric of distance between a record and a cluster centroid. If the attributes of the records are not quantitative but categorical in nature, such as Income Level with values {low, medium, high} or Married with values {yes, no} or State of Residence with values {Alabama, Alaska,…, Wyoming} then the distance metric is not meaningful. Define a more suitable similarity metric that can be used for clustering data records that contain categorical data.

Answer:

We can define a distance metric or rather a similarity metric between two records based on the number of common values the two records have across all dimensions.

For n dimensional data records, we define the similarity of two records rj and rk as

similarity(rj, rk) = sim(rj1, rk1) + sim(rj2, rk2) + ... + sim(rjn, rkn)

where sim(rji, rki) is 1 if rji = rki else 0.

In this case, higher similarity means the records are closer together with respect to the usual distance metric.

For example, consider the following 3 records:

RID	INCOME LEVEL	MARRIED	STATE
1	high	yes	ny
2	low	no	ny
3	high	yes	ca

We have the following similarity values:
similarity(1,2) = 1
similarity(1,3) = 2
similarity(2,3) = 0

Records 1 and 3 are the most similar (or closest).