CHAPTER 1: DATABASES AND DATABASE USERS

Answers to Selected Exercises

1.8 - Identify some informal queries and update operations 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.

1.9 - 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.

1.10 - Specify 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.

1.11 - 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.

1.12 – Cite some examples of integrity constraints that you think can apply to 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).

ScholarStock

1.13 - Give examples of systems in which it may make sense to use traditional file processing instead of a database approach.

Answer:

~~1.~~1. Small internal utility to locate files

~~2.~~2. Small single user application that does not require security (such as a customized calculator or a personal address and phone book)

~~3.~~3. Real-time navigation system (with heavy computation and very little data)

~~4.~~4. The students may think of others.

Formatted: Bullets and Numbering

1.14 - Consider Figure 1.2.

~~a.~~a. If the name of the 'CS' (Computer Science) Department changes to 'CSSE' (Computer Science and Software Engineering) Department and the corresponding prefix for the course number also changes, identify the columns in the database that would need to be updated.

~~b.~~b. Can you restructure the columns in COURSE, SECTION, and PREREQUISITE tables so that only one column will need to be updated?

Formatted: Bullets and Numbering

Answer:

a. The following columns will need to be updated.

Table	Column(s)
STUDENT	Major
COURSE	CourseNumber and Department
SECTION	CourseNumber
PREREQUISITE	CourseNumber and PrerequisiteNumber

b. You should split the following columns into two columns:

Table	Column	Split Columns

COURSE	CourseNumber	CourseDept and CourseNum
SECTION	CourseNumber	CourseDept and CourseNum
PREREQUISITE	CourseNumber	CourseDept and CourseNum
PREREQUISITE	PrerequisiteNumber	PreReqDept and PreReqNum

Note that in the COURSE table, the column CourseDept will not be needed after the above change, since it is redundant with the Department column.

CHAPTER 2: DATABASE SYSTEM CONCEPTS AND ARCHITECTURE

Answers to Selected Exercises

2.12 - 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.

2.13 - No solution provided.

2.14 - if you were designing a Web-based system to make airline reservations and to sell airline tickets, which DBMS Architecture would you choose from Section 2.5? Why? Why would the other architectures not be a good choice?

Answer:

2.5.4 Three-Tier Client/Server Architecture for Web Application is the best choice. The Client consists of Web User Interface. The Web Server contains the application logic which includes all the rules and regulations related to the reservation process and the issue of tickets; the Database Server contains the DBMS.

2.5.1 Centralized DBMS Architecture would not work since the user interface and database server are on different machines for a web-based system.

2.5.2 Basic Client/Server Architecture and 2.5.3 Two-Tier Client/Server Architecture would work if the Business Logic can reside on server other than the DBMS Server. In general, if the business

logic was on the DBMS Server, it will put an excessive burden on the server. If the business logic were to reside on the web client, it will burden the communication network as well a possibly thin client.

2.15 - Consider Figure 2.1. In addition to constraints relating the values of columns in one table to columns in another table, there are also constraints that impose restrictions on values in a column or a combination of columns within a table. One such constraint forces that a column or a group of columns must be unique across all rows in the table. For example, in the STUDENT table, the StudentNumber column must be unique (to prevent two different students from having the same StudentNumber). Identify the column or the group of columns in the other tables that must be unique across all rows in the table?

Answer:

Table	Column(s)
COURSE	CourseNumber Since this contains the combination of the department and the number that must be unique within the department. Note we will overlook the fact this does not accommodate a department from offering several "Special Topics" course with the same CourseNumber but different titles. We could make this a combination of CourseNumber and CourseName, but this is more succeptible to someone mistyping while entering data.
PREREQUISITE	The combination of CourseNumber and PrerequisiteNumber
SECTION	SectionIdentifier We assume that no two sections can have the same SectionIdentifier. If we were to consider that SectionIdentifier is unique only within a given course offered in a given term (such as section 2 of CS101) then the answer changes to the combination of SectionIdentifier, CourseNumber, Semester, and Year.
GRADE_REPORT	StudentNumber and SectionIdentifier As per assumption stated in SECTION, the SectionIdentifier will be different if a student takes the same course or a different course in another term.

CHAPTER 7(3): DATA MODELING USING THE ENTITY-RELATIONSHIP (ER) MODEL

Answers to Selected Exercises

7.16 - Consider the following set of requirements for a UNIVERSITY database that is used to

keep track of students' transcripts. This is similar but not identical to the database shown in Figure 1.2:

(a) The university keeps track of each student's name, student number, social security number, current address and phone, 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,

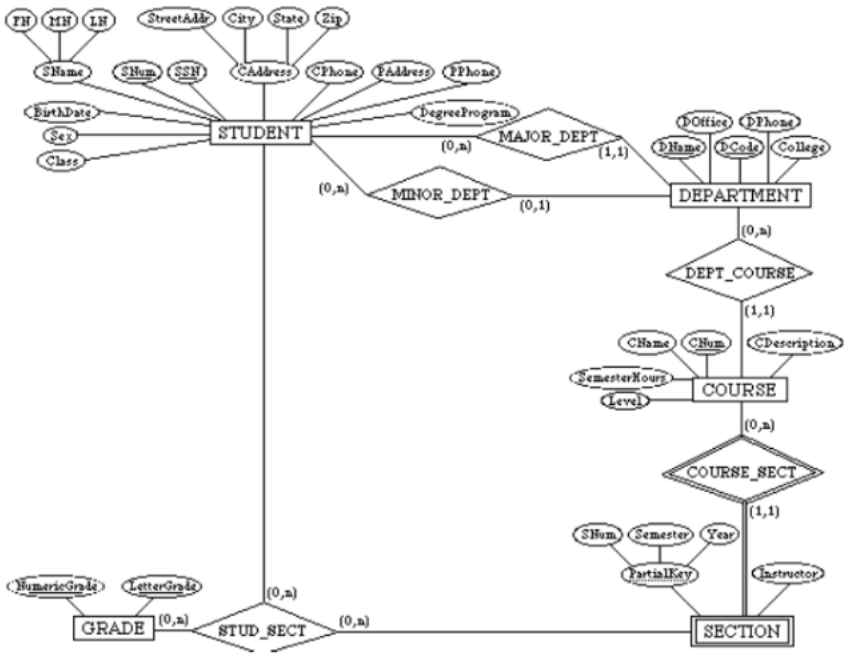
4 for F, D, C, B, A, respectively).

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

ScholarStock

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

7.16

7.17 - 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

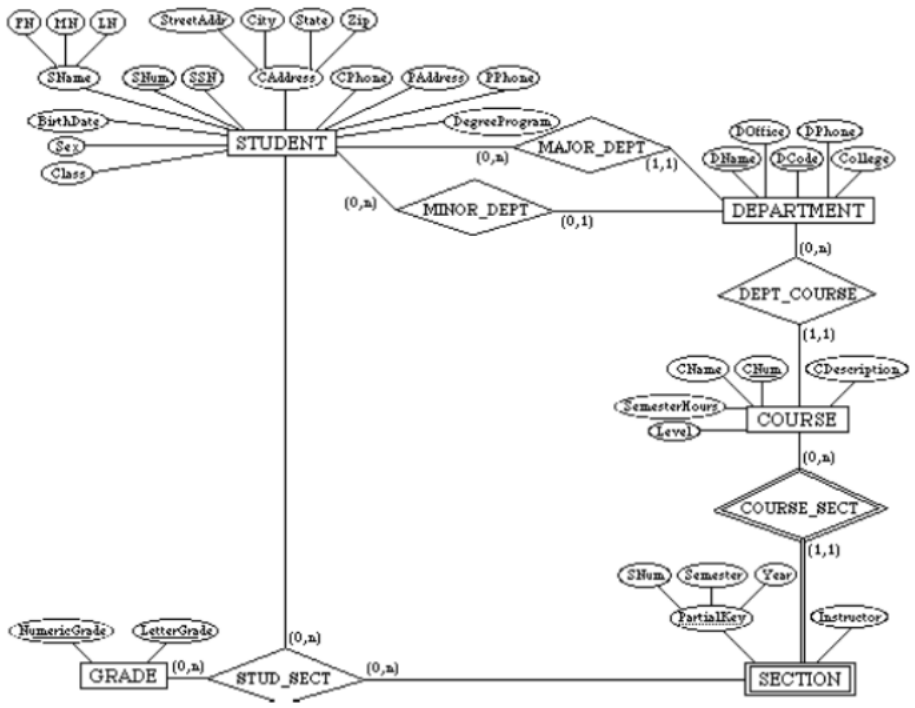
course name, semester, year, and grade. Design an attribute to hold this information. Use the conventions of Figure 7.5.

Answer:

{ PreviousEducation (CollegeName, StartDate, EndDate,

{ Degree (DegreeName, Month, Year) },

{ Transcript (CourseName, Semester, Year, Grade) }) }

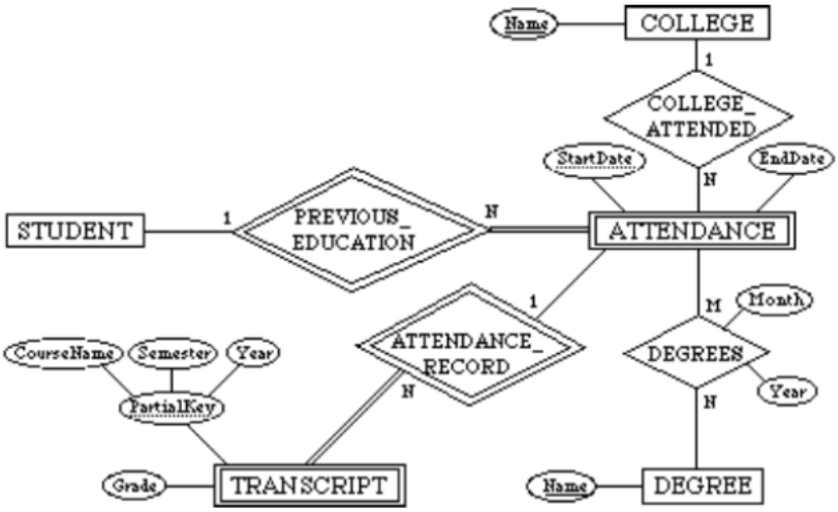


ER Schema Diagram for Exercise 3.16 7.16

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

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.



7.19 - Consider the ER diagram of Figure 7.20, 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.

7.20 - No solution provided.

7.21 -

Additional information:

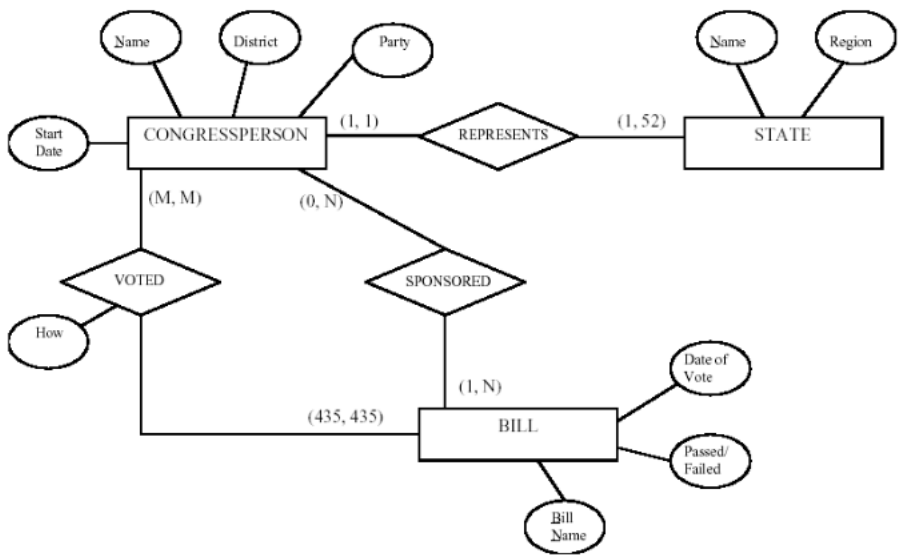
ScholarStock

- 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.



7.22 - 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 7.8):

Performance({Hitting(AtBat#, Inning#, HitType, Runs, RunsBattedIn, StolenBases)},

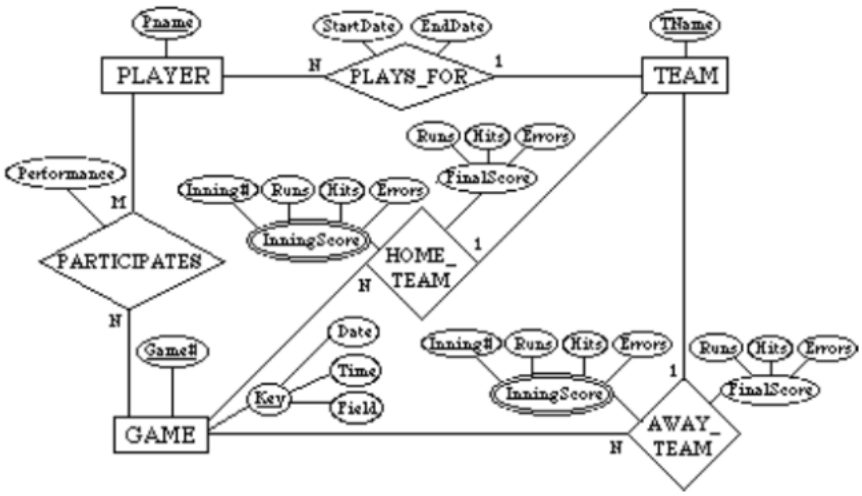
{Pitching(Inning#, Hits, Runs, EarnedRuns, StrikeOuts, Walks, Outs,

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.

ScholarStock



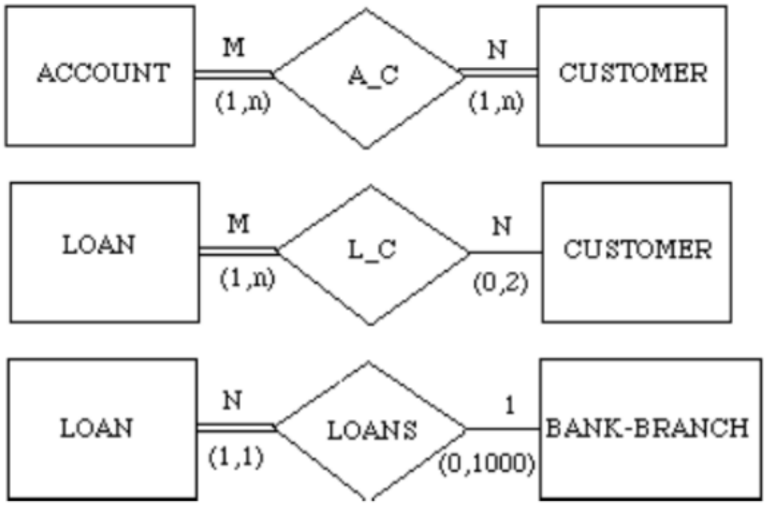
7.23 - Consider the ER diagram shown in Figure 7.21 for part of a BANK database. Each bank can have multiple branches, and each branch can have multiple accounts and loans.

(a) List the strong (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 ay 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:

ScholarStock

7.24 - Consider the ER diagram in Figure 7.22. Assume that an employee may work in up to two departments or may not be assigned to 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.

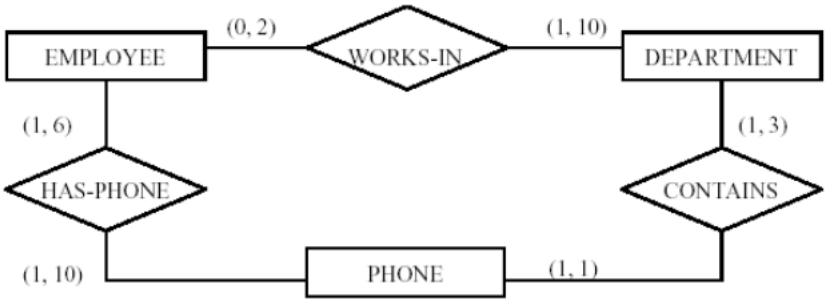
- Each phone is used by one, and only one, department.

- 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)

7.25 - Consider the ER diagram in Figure 7.23. 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,

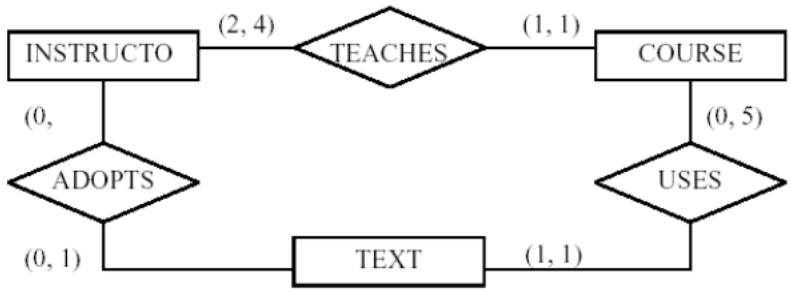
- ___hence it is adopted by some instructor who teaches that course.

- An instructor is considered to adopt a text if it is used in some course taught

- by that instructor.

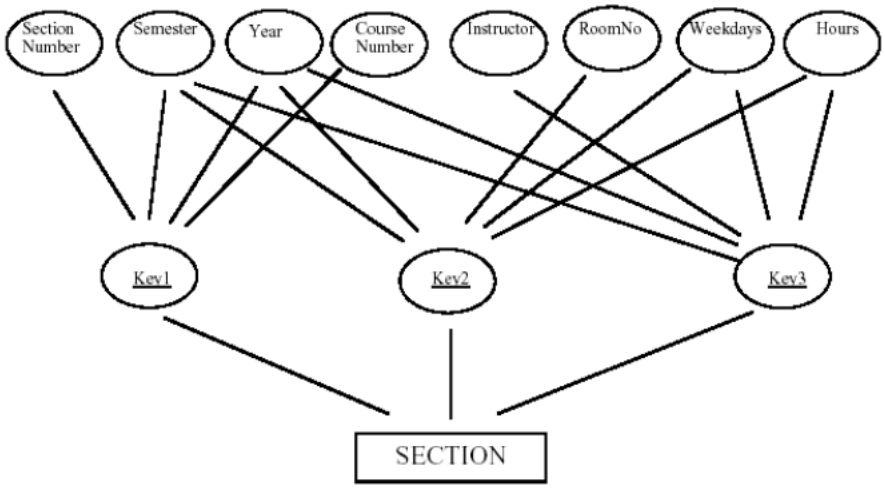
The resulting ER Diagram will have the (min, max) constraints shown in Figure B.

Figure B



7.26 - 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:



7.27 - Cardinality ratios often dictate the detailed design of a database. The cardinality ratio depends on the real-world meaning of the entity types involved and is defined by the specific application. For the binary relationships below, suggest cardinality ratios based on common-sense meaning of the entity types. Clearly state any assumptions you make.

	Entity 1	Cardinality Ratio	Entity 2
~~1.~~1.	Student		SocialSecurityCard
~~2.~~2.	Student		Teacher
~~3.~~3.	ClassRoom		Wall
~~4.~~4.	Country		CurrentPresident
~~5.~~5.	Course		TextBook
~~6.~~6.	Item (that can be found in an order)		Order
~~7.~~7.	Student		Class

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

~~8.~~8.	Class		Instructor
~~9.~~9.	Instructor		Office
~~10.~~10.	E-bay Auction item		E-bay bid

Answer:

	Entity 1	Cardinality Ratio	Entity 2
~~1.~~1.	Student	1-many A student may have more than one social security card (legally with the same unique social security number), and every social security number belongs to a unique student.	SocialSecurityCard
~~2.~~2.	Student	Many-many Generally students are taught by many teachers and a teacher teaches many students.	Teacher
~~3.~~3.	ClassRoom	Many-many Don't forget that the wall is usually shared by adjacent rooms.	Wall
~~4.~~4.	Country	1-1 Assuming a normal country under normal circumstances having one president at a time.	CurrentPresident

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

~~5.~~5.	Course	Many-many A course may have many textbooks and a text book may be prescribed for different courses.	TextBook
~~6.~~6.	Item (that can be found in an order)	Many-many Assuming the same item can appear in different orders.	Order
~~7.~~7.	Student	Many-many One student may take several classes. Every class usually has several students.	Class
~~8.~~8.	Class	Many-to-1 Assuming that every class has a unique instructor. In case instructors were allowed to team teach, this will be many-many.	Instructor
~~9.~~9.	Instructor	1-1 Assuming every instructor has only one office and it is not shared. In case of offices shared by 2 instructors, the relationship will be 2-1. Conversely, if an instructor has a joint appointment (in two departments) and offices in both departments, then the relationship will be 1-2. In a very general case, it may be many-many.	Office
~~10.~~10.	E-bay Auction item	1-many 1 item has many bids and	E-bay bid

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

Formatted: Bullets and Numbering

		a bid is unique to an item (assuming a regular auction format).	

7.28-7.29 – No solutions provided.

3.30 - Illustrate the UML Diagram for exercise 7.16. Your UML design should observe the following requirements:

~~a.~~a. The student should have the ability to compute his/her GPA and add or drop majors and minors.

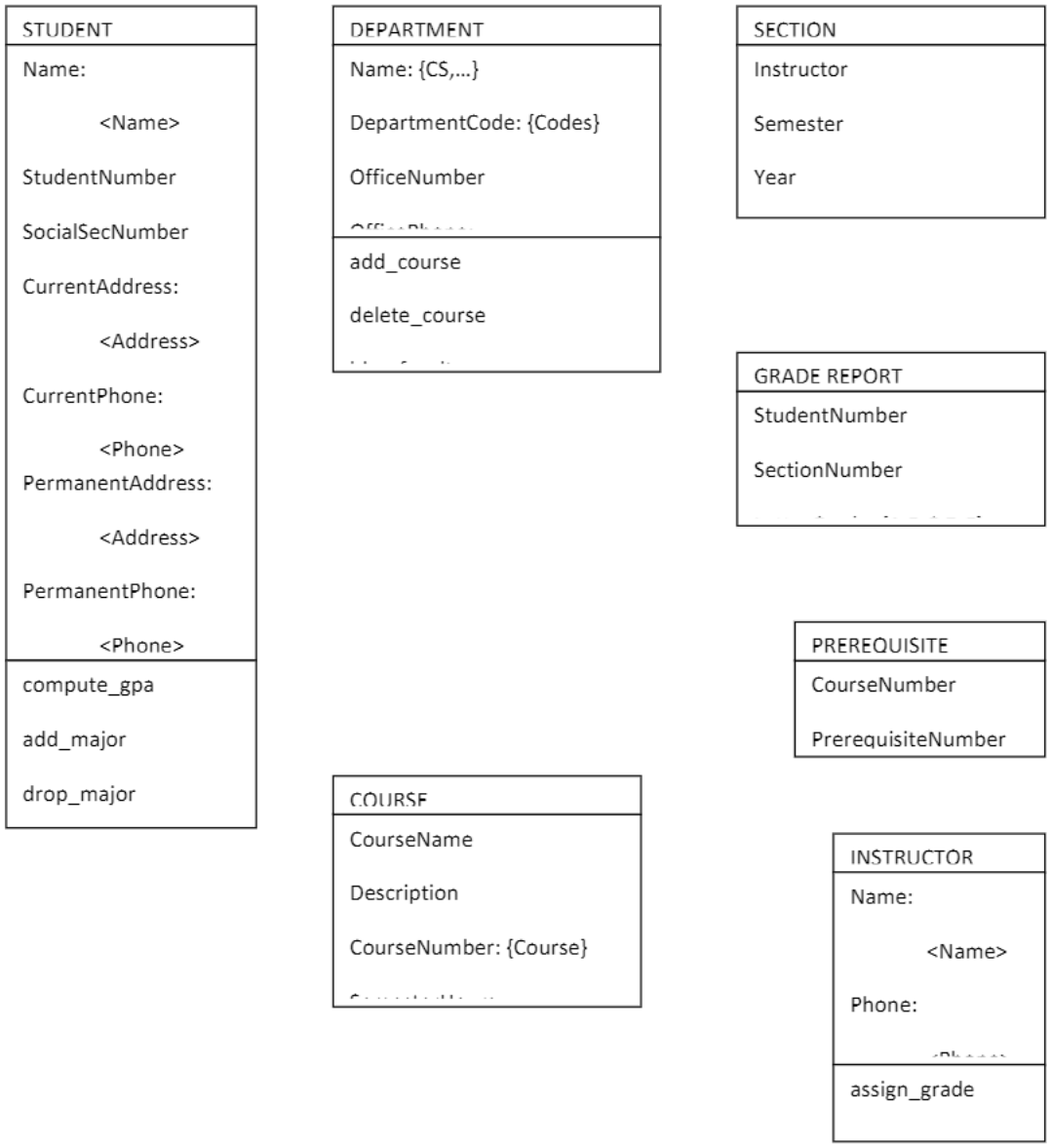
~~b.~~b. Each department should be to able add or delete courses and hire or terminate faculty.

~~c.~~c. Each instructor should be able to assign or change a grade to a student for a course.

Note: Some of these functions may be spread over multiple classes.

Answer:

Formatted: Bullets and Numbering



This one isn't finished yet. The relationships need to be added and some of the functions should be moved to span classes.

ScholarStock

CHAPTER 4(6): Basic SQL

Answers to Selected Exercises

4.5 - 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) );
```

4.6 - Repeat Exercise 4.5, but use the AIRLINE schema of Figure 3.8.

Answer:

The following referential integrity constraints should hold:

ScholarStock

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,
```

ScholarStock

```
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) );
```

ScholarStock

4.7 - Consider the LIBRARY relational database schema of Figure 4.6. Choose the appropriate action (reject, cascade, set to null, set to default) for each referential integrity constraint, both for the *deletion* of a referenced tuple, and for the *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)

4.8 - Write appropriate SQL DDL statements for declaring the LIBRARY relational database schema of Figure 4.6. Specify the keys and referential triggered actions.

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) );
```

4.9 - How can the key and foreign key constraints be enforced by the DBMS? Is the enforcement technique you suggest difficult to implement? Can the constraint checks be executed 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.

4.10 – No solution provided.

4.11 - Specify the updates of Exercise 3.11 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'.

ScholarStock

```
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
```

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

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

ScholarStock

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' )
```

4.13 - 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

4.14 – No solution provided

4.15 - Consider the EMPLOYEE table's constraint EMPSUPERFK as specified in Figure 4.2 is changed to read as follows:

CONSTRAINT EMPSUPERFK

FOREIGN KEY (SUPERSSN) REFERNCES EMPLOYEE(SSN)

ON DELETE CASCADE ON UPDATE CASCADE,

Answer the following questions:

~~a.~~a. What happens when the following command is run on the database state shown in Figure 5.6?

Formatted: Bullets and Numbering

DELETE EMPLOYEE WHERE LNAME = 'Borg'

~~b.~~b. Is it better to CASCADE or SET NULL in case of EMPSUPERFK constraint ON DELETE?

Formatted: Bullets and Numbering

Answers:

a) The James E. Borg entry is deleted from the table, and each employee with him as a supervisor is also (and their supervisees, and so on). In total, 8 rows are deleted and the table is empty.

b) It is better to SET NULL, since an employee is not fired (DELETED) when their supervisor is deleted. Instead, their SUPERSSN should be SET NULL so that they can later get a new supervisor.

4.16 - Write SQL statements to create a table EMPLOYEE_BACKUP backup of EMPLOYEE table shown in Figure 3.6.

Answer:

INSERT INTO EMPLOYEE_BACKUP VALUES (SELECT * FROM EMPLOYEE)

CHAPTER 5(7): More SQL: Complex Queries, Triggers, Views, and Schema Modification

Answers to Selected Exercises

5.5 - Specify the following additional queries on the database of Figure 3.5 in SQL. Show

the query results if applied to the database of Figure 3.6.

(a) For each department whose average employee salary is more than $30,000, 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 5.4a). 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

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

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

(b) 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, Major

FROM STUDENT

WHERE NOT EXISTS (SELECT *

FROM GRADE_REPORT

WHERE StudentNumber= STUDENT.StudentNumber AND NOT(Grade='A'))

(b) SELECT Name, Major

FROM STUDENT

WHERE NOT EXISTS (SELECT *

FROM GRADE_REPORT

WHERE StudentNumber= STUDENT.StudentNumber AND Grade='A')

5.7 - In SQL, specify the following queries on the database specified in Figure 3.5 using the concept of nested queries and the concepts described in this chapter.

a. Retrieve the names of all employees who work in the department that has the employee with the highest salary among all employees.

b. Retrieve the names of all employees whose supervisor's supervisor has '888665555' for Ssn.

c. Retrieve the names of employees who make at least $10,000 more than the employee who is paid the least in the company.

Answers:

a) SELECT LNAME FROM EMPLOYEE WHERE DNO =

(SELECT DNO FROM EMPLOYEE WHERE SALARY =

(SELECT MAX(SALARY) FROM EMPLOYEE))

b) SELECT LNAME FROM EMPLOYEE WHERE SUPERSSN IN
(SELECT SSN FROM EMPLOYEE WHERE SUPERSSN = '888665555')

c) SELECT LNAME FROM EMPLOYEE WHERE SALARY >= 10000 +
(SELECT MIN(SALARY) FROM EMPLOYEE)

5.8 – No solution provided.

5.9 – No solution provided.

CHAPTER 6(8): THE RELATIONAL ALGEBRA AND RELATIONAL CALCULUS

Answers to Selected Exercises

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

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

ScholarStock

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

(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.

ScholarStock

(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

6.17 – No solution provided.

6.18 - Consider the LIBRARY relational schema shown in Figure 6.14, which is used to keep track of books, borrowers, and book loans. Referential integrity constraints are shown as directed arcs in Figure 6.14, as in the notation of Figure 3.7. 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)

6.19 – 6.21: No solutions provided.

6.22 – Consider the two tables *T*1 and *T*2 shown in Figure 6.15. Show the results of the following operations:

Answers:

(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

10a 5

15 b 8

25 a 6

10b 6

25 c 3

10b 5

(f)

P Q R A B C

10 a 5 10 b 5

6.23 – No solution provided.

6.24 - Specify queries (a), (b), (c), (e), (f), (i), and (j) of Exercise 6.16 in both tuple and 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) }

ScholarStock

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 c, d, and e of Exercise 6.18 in both tuple and domain relational calculus.

Answer:

(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')) }

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)

6.29 – No solution provided.

6.30 - Show how you can 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) }

6.31 – No solution provided.

6.32 - A nested query is query within a query. More specifically, a nested query is a parenthesized query that can be used as a value in a number of places, such as instead of a relation or a selection condition. Specify the following queries on the database specified in Figure 3.5 using the concept of nested queries and the relational operators discussed in this chapter. Also show the result of each query as it would apply to the database state of Figure 3.6.

a. List the names of all employees who work in the department that has the employee with the highest salary among all employees.

b. List the names of all employees whose supervisor's supervisor has '888665555' for SSN.

c. List the names of employees who make at least $10,000 more than the employee who is paid the least in the company.

Answers:

a) RESULT <-- P LNAME (S DNO = (P DNO (S SALARY = MAX(SALARY) EMPLOYEE)) EMPLOYEE)

b) RESULT <-- P LNAME (S SUPERSSN = (P SSN (S SUPERSSN = '88866555' EMPLOYEE)) EMPLOYEE)

c) RESULT <-- P LNAME (S SALARY >= 10000 + P SALARY (S SALARY = MIN(SALARY) EMPLOYEE) EMPLOYEE)

6.33 - State whether the following conclusion are true or false:

a. NOT (P(x) OR Q(x)) ⇒ (NOT (P(x)) AND (NOT (Q(x)))

b. NOT $(\exists x)$ $(P(x)) \Rightarrow \forall x$ (NOT $(P(x))$

b. $(\exists x)$ $(P(x)) \Rightarrow \forall x$ $((P(x))$

Answers:

a. TRUE

b. TRUE

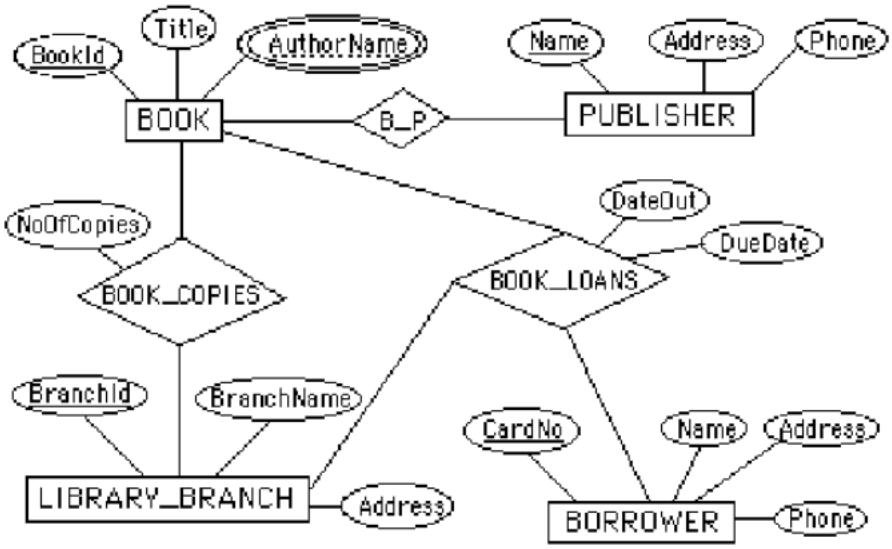
c. FALSE

CHAPTER 9(9): RELATIONAL DATABASE DESIGN BY ER- AND EER-TO-RELATIONAL MAPPING

Answers to Selected Exercises

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

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.

9.4 - Figure 9.8 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.

9.5 Map the BANK ER schema of Exercise 7.23 (shown in Figure 7.21) into a relational schema. Specify all primary keys and foreign keys. Repeat for the AIRLINE schema (Figure 7.20) of Exercise 7.19 and for the other schemas for Exercises 7.16 through 7.24.

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.

9.6 – 9.9: No solutions provided.

CHAPTER 15: Database Design Theory: Normalization Algorithms

Answers to Selected Exercises

15.17 - Show that the relation schemas produced by Algorithm 15.4 are in 3NF.

Answer:

We give a proof by contradiction. Suppose that one of the relations R i resulting from Algorithm 15.1 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.

15.18 - Show that, if the matrix S resulting from Algorithm 15.3 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).

15.19 - Show that the relation schemas produced by Algorithm 15.5 are in BCNF.

Answer:

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

15.20 – No Solution Provided

15.21 - Specify a template dependency for join dependencies.

Answer:

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

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

	a	b	c_1
hypothesis	a	b_1	c
	a_1	b	c
conclusion	a	b	c

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

15.22 - Specify all the inclusion dependencies for the relational schema of Figure 3.5.

Answer:

The inclusion dependencies will correspond to the foreign keys shown in Figure 3.7.

15.23 - Prove that a functional dependency satisfies the formal definition of multi-valued 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.

15.24 - 15.31: No solutions provided.

15.32 - 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 15.2.4)

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)

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

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

$R1 \cap R2 = M$
$R2 - R1 = \{MP, C\}$

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

CHAPTER 17(16): DISK STORAGE, BASIC FILE STRUCTURES, AND HASHING

Answers to Selected Exercises

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

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

(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

ScholarStock

(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)

17.28 - A file has r=20,000 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 17.27.

(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 that only 80% of the STUDENT records from Exercise 17.28 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-of-record 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))

17.30 - 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.

ScholarStock

(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

17.31 - 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

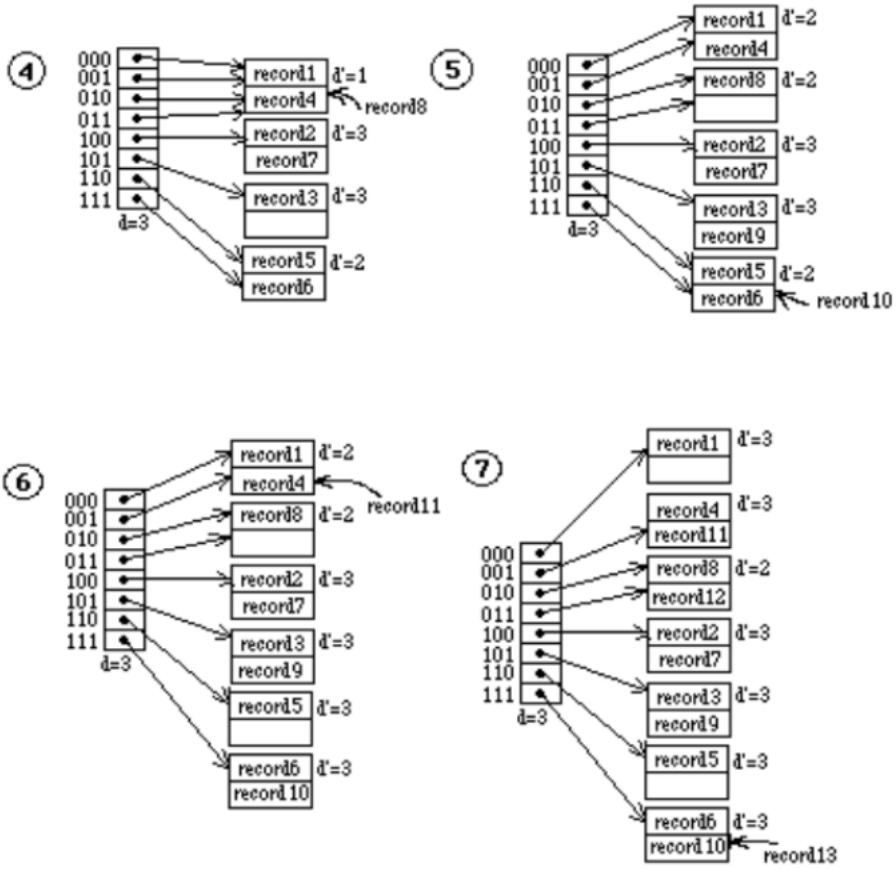
17.32 - Load the records of Exercise 17.31 into expandable hash files based on extendible hashing. Show the structure of the directory at each step. Show the directory at each step, and the global and local depths. Use the hash function h(k) = K mod 128.

Answer:

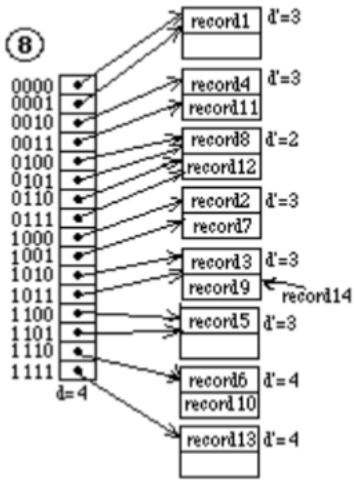
Hashing the records gives the following result:

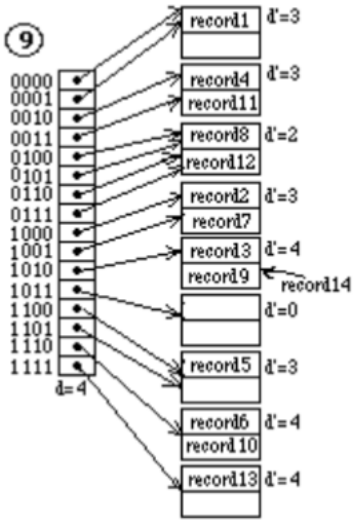
	K	h(K) (bucket number)	binary h(K)
record1	2369	1	00001
record2	3760	16	10000
record3	4692	20	10100
record4	4871	7	00111
record5	5659	27	11011
record6	1821	29	11101
record7	1074	18	10010
record8	7115	11	01011
record9	1620	20	10100
record10	2428	28	11100
record11	3943	7	00111
record12	4750	14	01110
record13	6975	31	11111
record14	4981	21	10101
record15	9208	24	11000

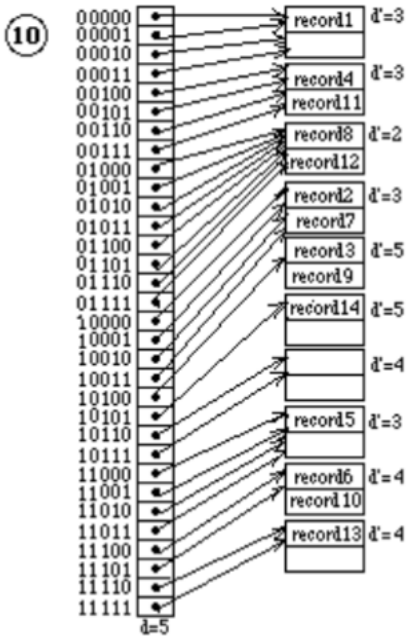
Extendible hashing:



ScholarStock



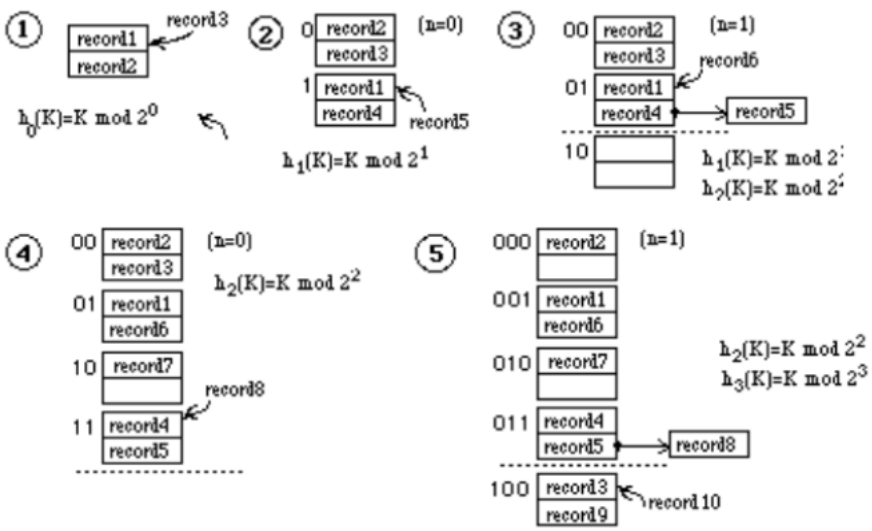


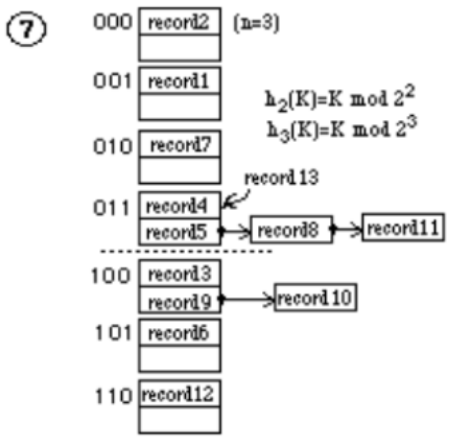


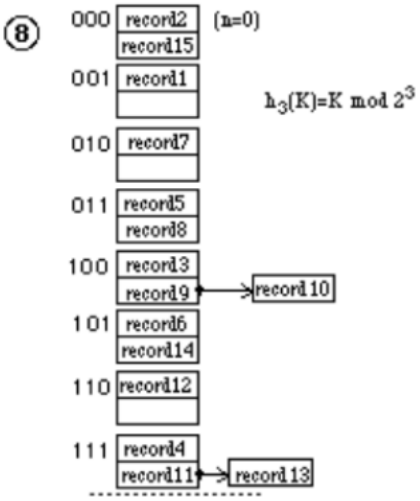
ScholarStock

17.33 - Load the records of Exercise 17.31 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 triggered. It is also possible to merge brackets in the reverse order in which they were created; a merge operation would be triggered whenever lf becomes less than another threshold, say 0.6.

17.34 – 17.36: No solutions provided

17.37 - 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.

17.38 - No solution provided.

17.39 - 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.

17.40 – 17.41: No solutions provided.

17.42 - Suppose that a file initially contains r=120,000 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 240,000.

(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?

Answer:

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.

17.43 - 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.

Answer:

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.

17.44 - 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.

17.45 - 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 112,000 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 -