Homework-3

Chapter 4 Intermediate SQL(Pg-152)

Name: ABID ALI

Roll : 2019380141

Question No. 4.1

Write the following queries in SQL:
a. Display a list of all instructors, showing their ID, name, and the number of sections that they have taught. Make sure to show the number of sections as 0 for instructors who have not taught any section. Your query should use an outerjoin, and should not use scalar subqueries.

b. Write the same query as above, but using a scalar subquery, without outerjoin.

c. Display the list of all course sections offered in Spring 2010, along with the names of the instructors teaching the section. If a section has more than one instructor, it should appear as many times in the result as it has instructors. If it does not have any instructor, it should still appear in the result with the instructor name set to "—".

d. Display the list of all departments, with the total number of instructors in each department, without using scalar subqueries. Make sure to correctly handle departments with no instructors.

Answer No. 4.1

a)

select ID, name, count(course_id, section_id, year,semester) as 'Number of sections' from instructor natural left outer join teaches group by ID, name;

b)

select ID, name, (select count(*) as ’Number of sections’ from teaches T where T.id = I.id) from instructor I;

c)

select course_id, section_id, ID, decode(name, NULL, ’−’, name) from (section natural left outer join teaches) natural left outer join instructor where semester=’Spring’ and year= 2010;

d)

select dept_name, count(ID) from department natural left outer join instructor group by dept_name;

Question No. 4.5

Show how to define the view *student grades* (*ID, GPA*) giving the gradepoint average of each student, based on the query in Exercise 3.2; recall that we used a relation *grade points*(*grade*, *points*) to get the numeric points associated with a letter grade. Make sure your view definition correctly handles the case of *null* values for the *grade* attribute of the *takes* relation.

Answer No. 4.5

create view student_grades(ID, GPA) as select ID, credit_points / decode(credit_sum, 0, NULL, credit _sum) from ((select ID, sum(decode(grade, NULL, 0, credits)) as credit_sum, sum(decode(grade, NULL, 0, credits*points)) as credit_points from(takes natural join course) natural left outer join grade_points group by ID) union select ID, NULL from student where ID not in (select ID from takes));

Question No. 4.6

Complete the SQL DDL definition of the university database of Figure 4.8 to include the relations *student*, *takes*, *advisor*, and *prereq*.

employee (*employee name*, *street*, *city*)
works (*employee name*, *company name*, *salary*)
company (*company name*, *city*)
manages (*employee name*, *manager name*)

Figure 4.11 Employee database for Figure 4.7 and 4.12.

Answer No. 4.6

We would execute the following SQL commands one after another in the given order:

i) create table student (ID varchar (5), name varchar (20) not null, dept_name varchar (20), tot_cred numeric (3,0) check (tot_cred >= 0), primary key (ID), foreign key (dept_name) references department on delete set null);

ii) create table takes (ID varchar (5), course_id varchar (8), section_id varchar (8), semester varchar (6), year numeric (4,0), grade varchar (2), primary key (ID, course_id, section_id, semester, year), foreign key (course_id, section_id, semester, year) references section on delete cascade, foreign key (ID) references student on delete cascade);

iii) create table advisor (i_id varchar (5), s_id varchar (5), primary key (s_ID), foreign key (i_ID) references instructor (ID) on delete set null, foreign key (s_ID) references student (ID) on delete cascade);

iv) create table prereq (course_id varchar(8), prereq_id varchar(8), primary key (course_id, prereq_id), foreign key (course_id) references course on delete cascade, foreign key (prereq_id) references course);

Question No. 4.8

As discussed in Section 4.4.7, we expect the constraint "an instructor cannot teach sections in two different classrooms in a semester in the same time slot" to hold.

a. Write an SQL query that returns all (*instructor*, *section*) combinations that violate this constraint.

b. Write an SQL assertion to enforce this constraint (as discussed in Section 4.4.7, current generation database systems do not support such assertions, although they are part of the SQL standard).

Answer No. 4.8

a) select ID, name, section_id, semester, year, time_slot_id, count(distinct building, room_number) from instructor natural join teaches natural join section group by (ID, name, section_id, semester, year, time_slot_id) having count(building, room_number) > 1;

b) create assertion check not exists (select ID, name, section_id, semester, year, time_slot_id, count(distinct building, room_number) from instructor natural join teaches natural join section group by (ID, name, section_id, semester, year, time_slot_id) having count(building, room_number) > 1);

Question No. 4.9

SQL allows a foreign-key dependency to refer to the same relation, as in the following example:

create table manager

(employee name **varchar**(20) **not null**

manager name **varchar**(20) **not null**,

primary key employee name,

foreign key (manager name) **references** manager **on delete cascade**)

Here, *employee name* is a key to the table *manager*, meaning that each employee has atmost one manager. The foreign-key clause requires that every manager also be an employee. Explain exactly what happens when a tuple in the relation *manager* is deleted.

Answer No. 4.9

The tuples of all employees of the management, at all levels, are also erased.

This occurs in a sequence of stages.

The first deletion will result in the deletion of all the tuples relating to the manager's direct employees. These deletions will result in the deletion of second level employee tuples, and so on, until all direct and indirect employee tuples have been removed.

Question No. 4.12

For the database of Figure 4.11, write a query to find those employees with no manager. Note that an employee may simply have no manager listed or may have a *null* manager. Write your query using an outer join and then write it again using no outer join at all.

Answer No. 4.12

a) select employee_name from employee natural left outer join manages where manager_name is null;

b) select employee_name from employee e where not exists (select employee_name from manages m where e.employee_name = m.employee_name and m.manager_name is not null);

Question No. 4.16

Referential-integrity constraints as defined in this chapter involve exactly two relations. Consider a database that includes the relations shown in Figure 4.12. Suppose thatwewish to require that every name that appears in *address* appears in either *salaried worker* or *hourly worker*, but not necessarily in both.

Answer No. 4.16

a) We offer a simplified version of the SQL syntax.

As part of the create table expression for address we include

foreign key (name) references salaried_worker or hourly_worker

b) To enforce this constraint, whenever a tuple is inserted into the address relation, a lookup on the name value must be made on the salaried_worker_relation and (if that lookup failed) on the hourly_worker relation (or vice-versa).

Question No. 4.18

Suppose user *A*, who has all authorizations on a relation *r* , grants select on relation *r* to **public** with grant option. Suppose user *B* then grants select on *r* to *A*. Does this cause a cycle in the authorization graph? Explain why.

Answer No. 4.18

Yes, it does cause a cycle in the authorization graph.

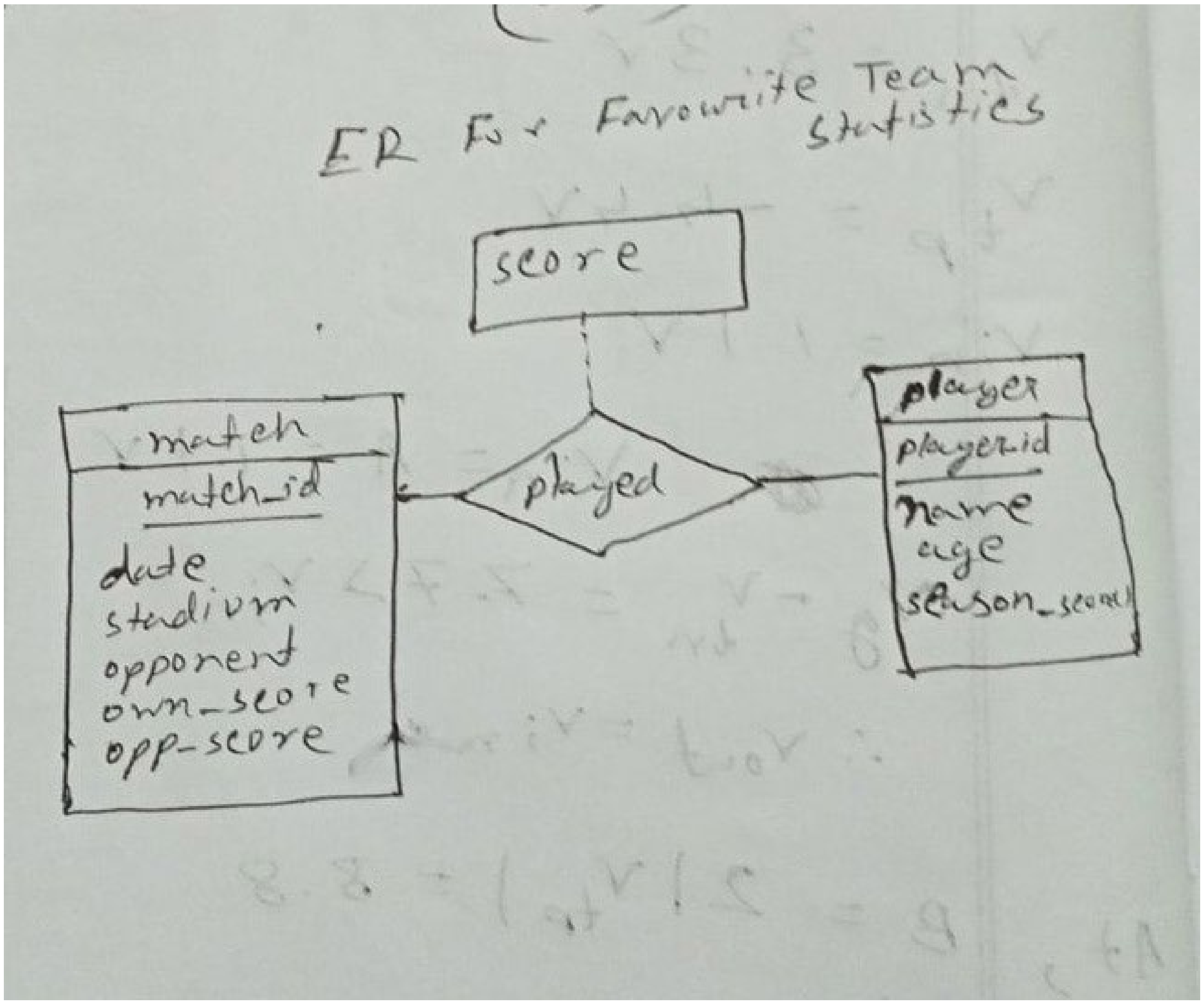
The grant to public results in an edge from A to public. The grant to the public operator provides authorization to everyone, B is now authorized. For each privilege granted to public, an edge must therefore be placed between public and all users in the system. If this is not done, then the user will not have a path from the root (DBA). And given the with grant option, B can grant select on r to A result in in an edge from B to A in the authorization graph. Thus, there is now a cycle from A to public, from public to B, and from B back to A.

Chapter 7 Database Design and the E-R Model(Pg-315)

Question No. 7.3

Design an E-R diagram for keeping track of the exploits of your favorite sports team. You should store the matches played, the scores in each match, the players in each match, and individual player statistics for each match. Summary statistics should be modeled as derived attributes.

Answer No. 7.3



Question No. 7.16

Construct appropriate relation schemas for each of the E-R diagrams inPractice Exercises 7.1 to 7.3.

Answer No. 7.16

a) {NOTE : the customer_id ,license_no , report_id is undelined in specific cases , thus is not showing "_" in their names in the PDF file}

customer (customer_id, name, address)

car (license, model)

owns(customer_id, license_no)

accident (report_id, date, place)

participated(license_no, report_id) policy(policy_id)

covers(policy_id, license_no)

premium_payment(policy_id, payment_no, due_date, amount, received_on)

b) {NOTE : the student_id , course_id, section_id,exam_id is undelined in specific cases , thus is not showing "_" in their names in the PDF file}

Student Exam tables:

i. Ternary Relationship:

student (student_id, name, dept_name, tot_cred)

course(course_id, title, credits)

section(course_id, section_id, semester, year)

exam(exam_id, name, place, time)

exam_marks(student_id, course_id, section_id, semester, year, exam_id, marks)

ii. Binary relationship:

student (<u>ID</u>, name, dept_name, tot_cred)

course(<u>course_id</u>, title, credits)

section(<u>course_id</u>, <u>section_id</u>, <u>semester</u>, <u>year</u>)

exam marks(<u>student_id</u>, <u>course_id</u>, <u>sec_id</u>, <u>semester</u>, <u>year</u>, <u>exam_id</u>, marks)

c) Player Match tables:

match(<u>match_id</u>, date, stadium, opponent, own_score, opp_score)

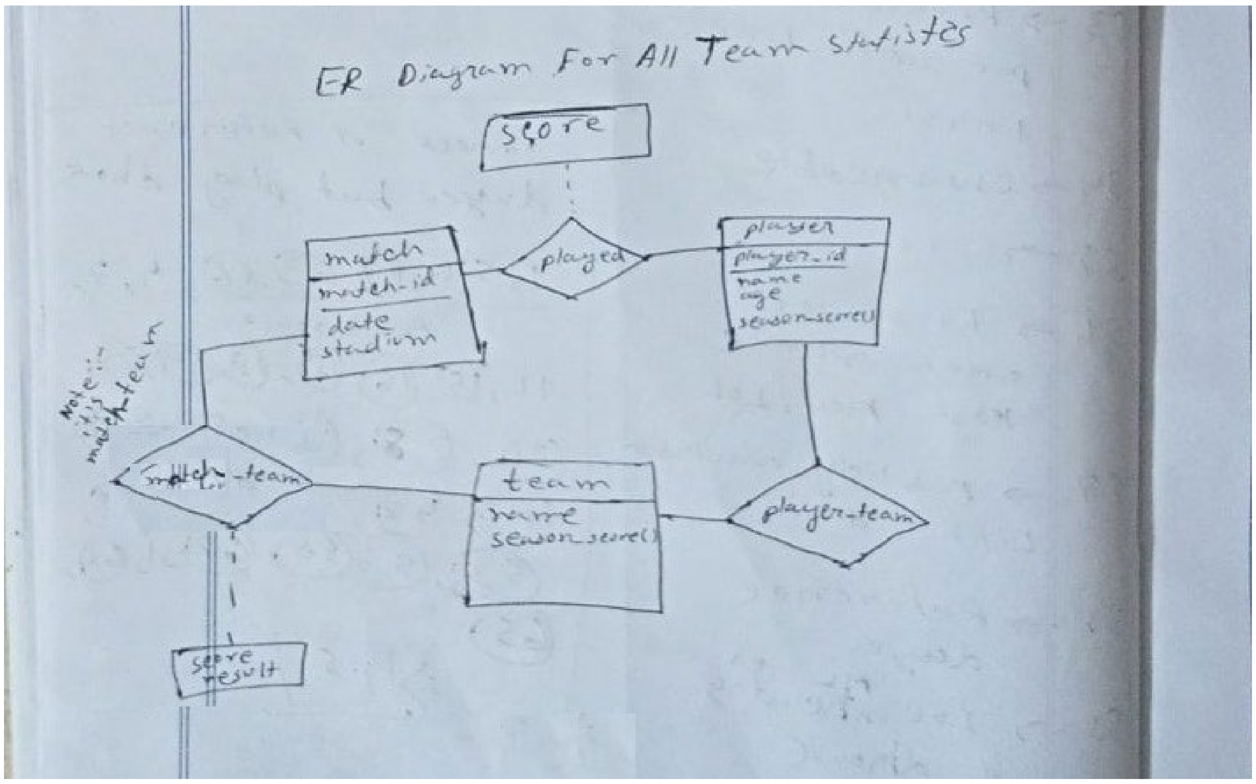
player(<u>player_id</u>, name, age, season_score)

played(<u>match_id</u>,<u>player_id</u>, score)

Question No. 7.17

Extend the E-R diagram of Practice Exercise 7.3 to track the same information for all teams in a league.

Answer No. 7.17



Question No. 7.20

Consider the E-R diagram in Figure 7.29, which models an online bookstore.

a. List the entity sets and their primary keys.

b. Suppose the bookstore adds Blu-ray discs and downloadable video to its collection. The same item may be present in one or both formats, with differing prices. Extend the E-R diagram to model this addition, ignoring the effect on shopping baskets.

c. Now extend the E-R diagram, using generalization, to model the case where a shopping basket may contain any combination of books, Blu-ray discs, or downloadable video.

Answer No. 7.20

a) Entity sets:

author(<u>name</u>, address, URL)

publisher(<u>name</u>, address, phone, URL)

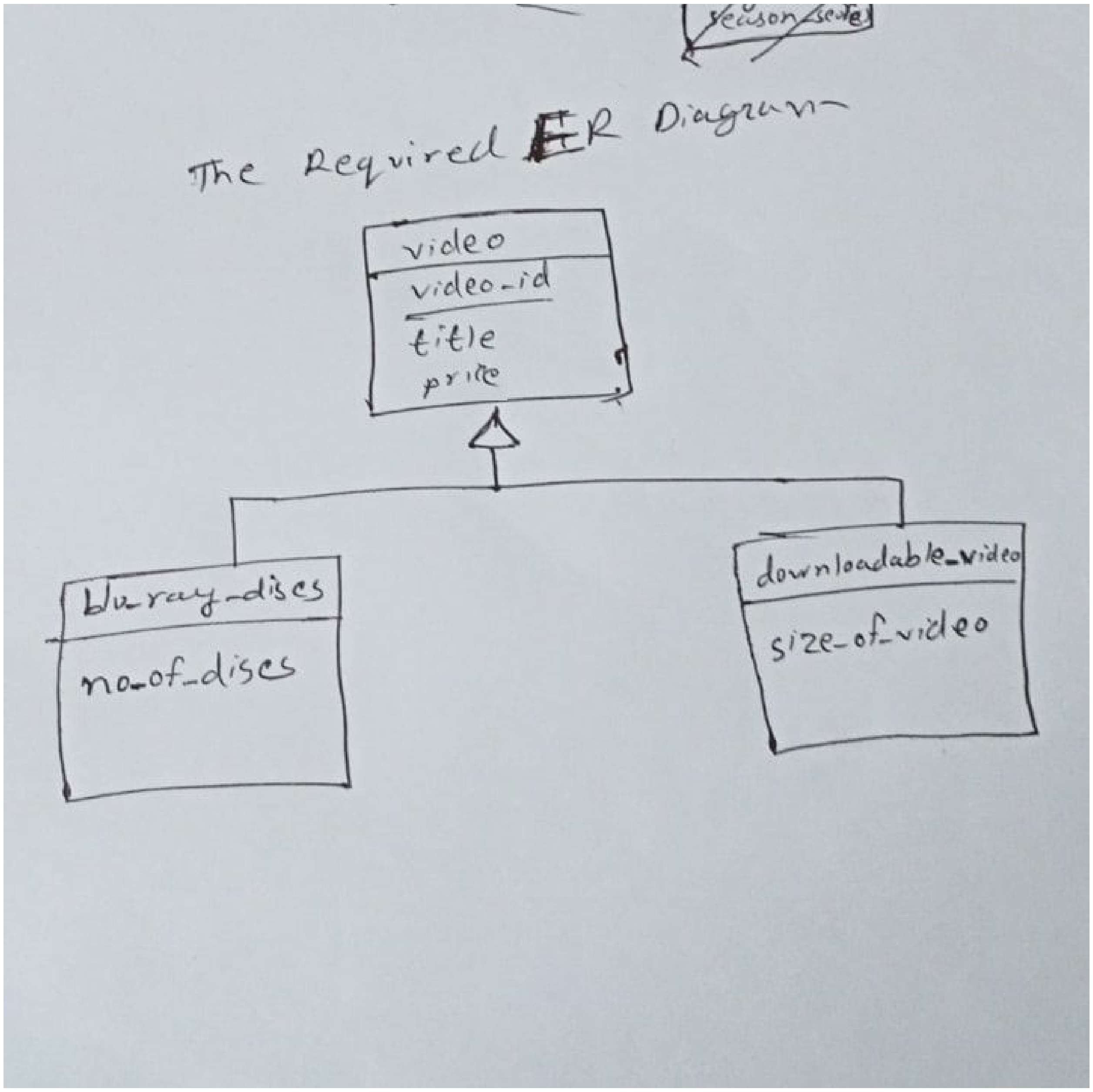
book(<u>ISBN</u>, title, year, price)

customer(<u>email</u>, name, address, phone)

shopping_basket(<u>basket_id</u>)

warehouse(<u>code</u>, address, phone)

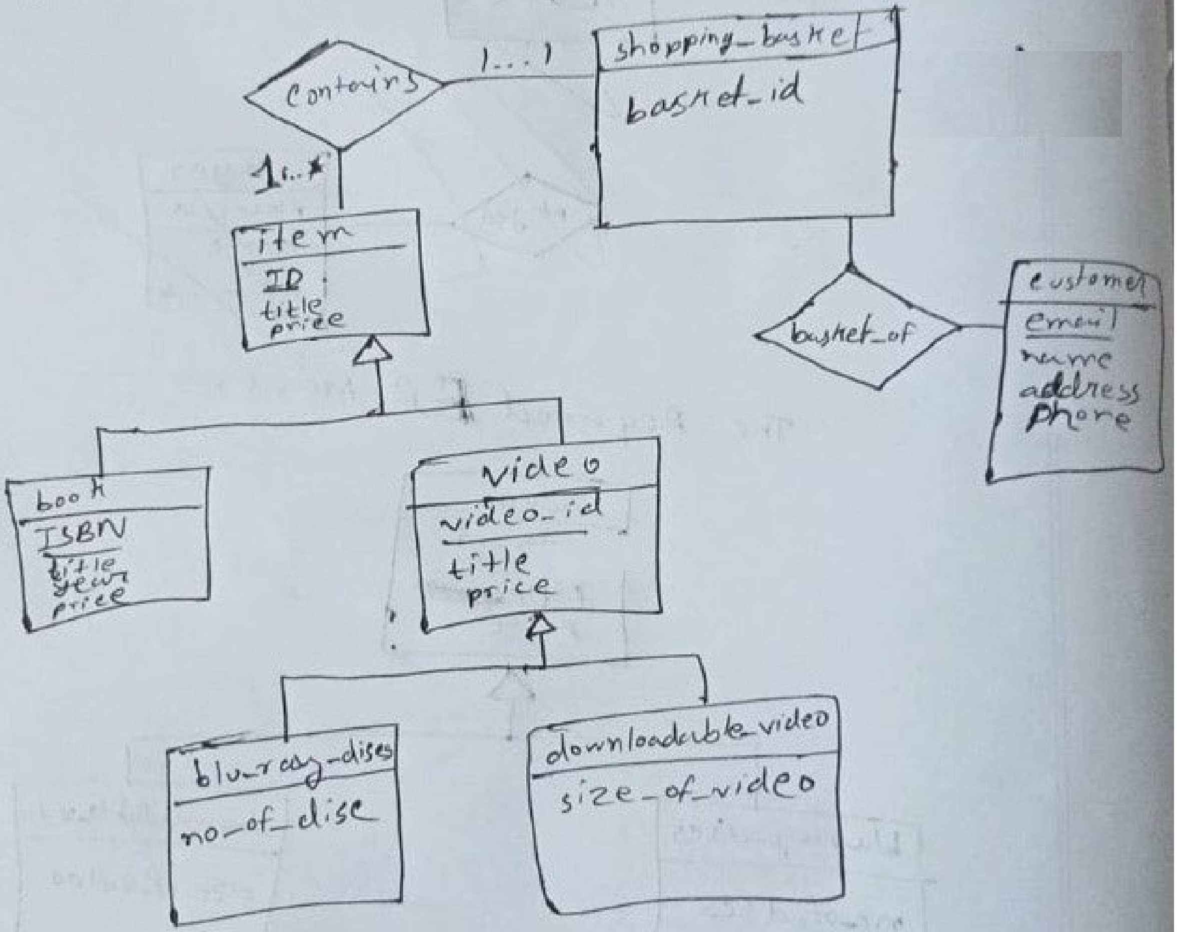
b)



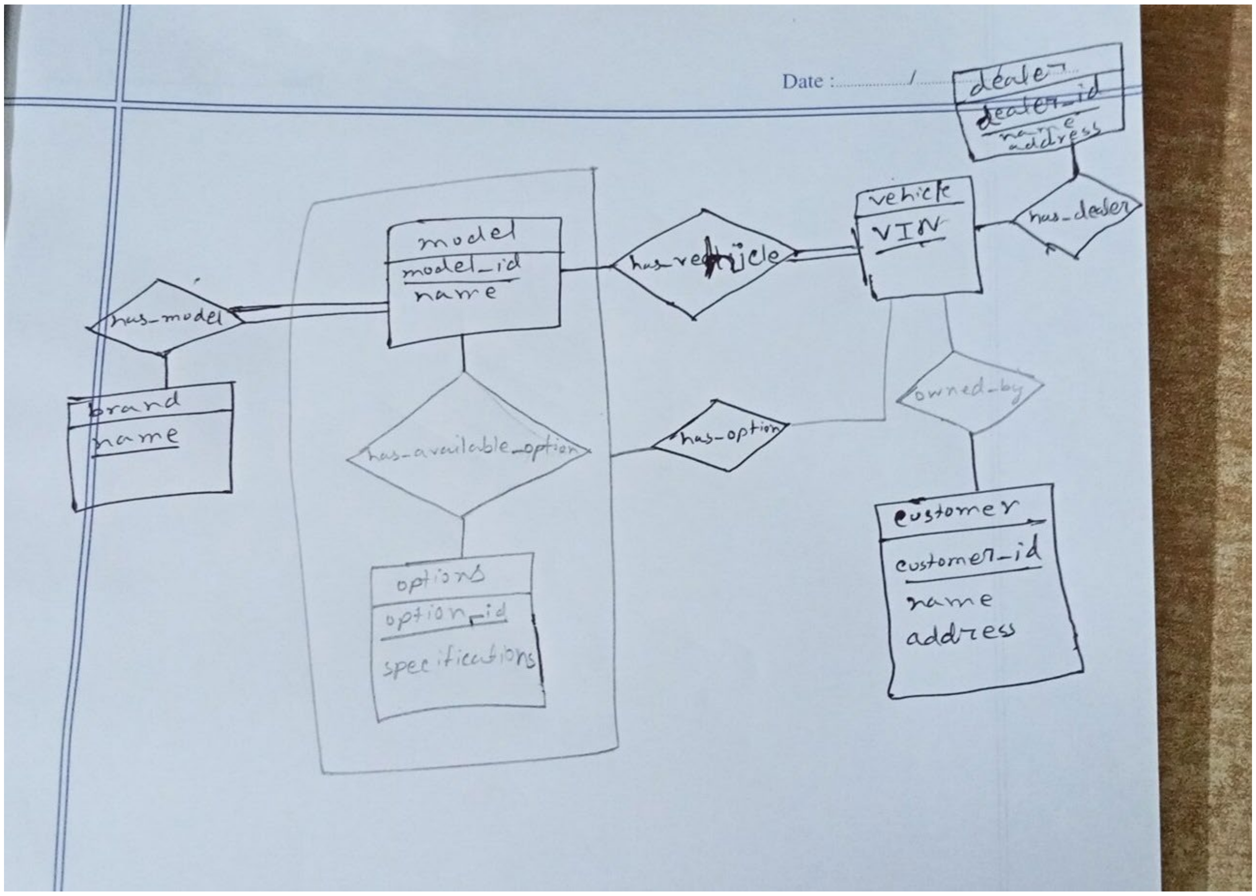
The Required ER Diagram
video
video-id
title
price
blu-ray-discs
no-of-discs
downloadable_video
size-of-video

c)

We consider that the ER in the given question to remain the same & we add modifications as required thus we get; the final ER :- (Note unchanged parts of the given ER of the question is not shown)



7.21



The E-R diagram is shown in above figure. Note that the has option relationship links a vehicle to an aggregation on the relationship has available option, instead of directly to the entity set options, to ensure that a particular vehicle instance cannot get an option that does not correspond to its model. The alternative of directly linking to options is acceptable if ensuring the above integrity constraint is not critical.

The relational schema, along with primary-key and foreign-key constraints is shown below.

{NOTE : some of the names, for example model_id,option_id… are undelined in specific cases, thus is not showing "_" in their names in the PDF file}

brand(<u>name</u>)

model(<u>model id,</u> name)

vehicle(<u>VIN</u>)

option(<u>option id</u>, specification)

customer(customer_id, name, address)

dealer(dealer_id, name, address)

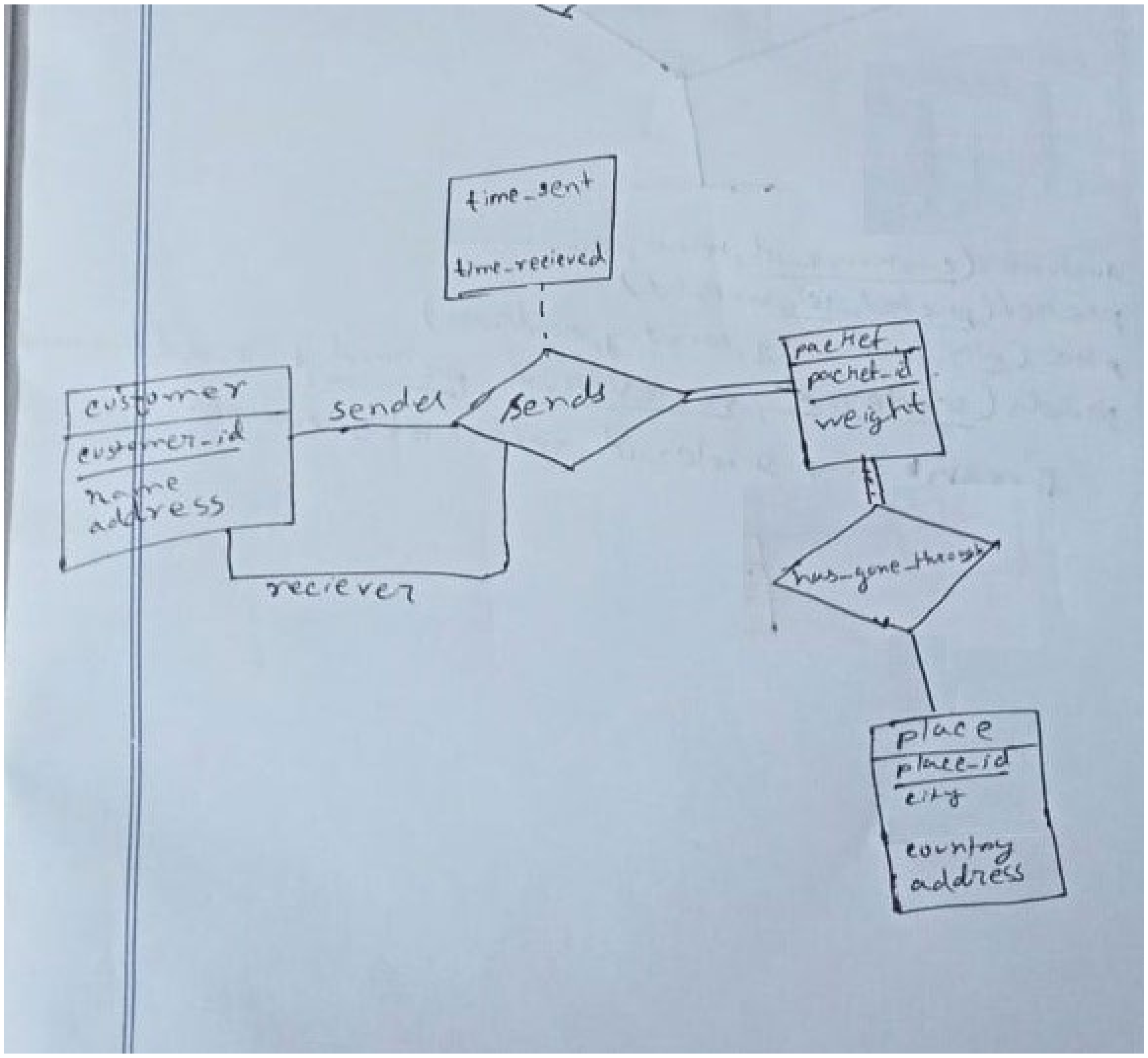
has_models(name, model_id , foreign key name references brand , foreign key model_id references model)

has_vehicles(model_id, VIN, foreign key VIN references vehicle, foreign key model_id references model)
available_options(model_id, option_id, foreign key option_id references option, foreign key model_id references model)

has_options(VIN, model_id, option_id, foreign key VIN references vehicle, foreign key (model_id, option_id) references available_options)

has dealer(VIN, dealer_id , foreign key dealer_id references dealer, foreign key VIN references vehicle)
owned by(VIN, customer_id, foreign key customer_id references customer, foreign key VIN references vehicle)

7.22



The above figure shows the required ER Diagram

The relational schema, including primary-key and foreign-key constraints, corresponding to the diagram is:

{NOTE : some of the names, for example customer_id,pacekt_id,… are undelined in specific cases, thus is not showing "_" in their names in the PDF file}

customer(<u>customer_id</u>, name, address)

packet(<u>packet_id</u>, weight)

place(<u>place_id</u>, city, country, address) sends(sender_id, receiver_id, packet_id, time_sent, time_received foreign key sender_id references customer, foreign key receiver_id references customer, foreign key packet_id references packet)

has_gone_through(packet_id, place_id

foreign key packet_id references packet, foreign key place_id references place)