CHAPTER 7

Database Design and the E-R Model

This chapter introduces the entity-relationship model in detail. A significant change in the 6th edition is the change in the E-R notation used in the book. There are several alternative E-R notations used in the industry. Increasingly, however, the UML class diagram notation is used instead of the traditional E-R notation used in earlier editions of the book. Among the reasons is the wide availability of UML tools, as well as the conciseness of the UML notation compared to the notation with separate ovals to represent attributes. In keeping with this trend, we have changed our E-R notation to be more compatible with UML.

The chapter covers numerous features of the model, several of which can be omitted depending on the planned coverage of the course. Extended E-R features (Section 7.8) and all subsequent sections may be omitted in case of time constraints in the course, without compromising the students understanding of basic E-R modeling. However, we recommend covering specialization (Section 7.8.1) at least in some detail, since it is widely used in object-oriented modeling.

The E-R model itself and E-R diagrams are used often in the text. It is important that students become comfortable with them. The E-R model is an excellent context for the introduction of students to the complexity of database design. For a given enterprise there are often a wide variety of E-R designs. Although some choices are arbitrary, it is often the case that one design is inherently superior to another. Several of the exercises illustrate this point. The evaluation of the goodness of an E-R design requires an understanding of the enterprise being modeled and the applications to be run. It is often possible to lead students into a debate of the relative merits of competing designs and thus illustrate by example that understanding the application is often the hardest part of database design.

Among the points that are worth discussing when coming up with an E-R design are:

1. Naming of attributes: this is a key aspect of a good design. One approach to design ensures that no two attributes share a name by accident; thus, if ID appears as an attribute of person, it should not appear as an attribute of

another relation, unless it references the ID of person. When natural joins are used in queries, this approach avoids accidental equation of attributes to some extent, although not always; for example, students and instructors share attributes ID and name (presumably inherited from a generalization *person*), so a query that joins the student and instructor relations would equate the respective attribute names.

2. Primary keys: one approach to design creates identifier values for every entity, which are internal to the system and not normally made visible to end users. These internal values are often declared in SQL as **auto increment**, meaning that whenever a tuple is inserted to the relation, a unique value is given to the attribute automatically.
   In contrast, the alternative approach, which we have used in this book, avoids creating artificial internal identifiers, and instead uses externally visible attributes as primary key values wherever possible.
   As an example, in any university employees and students have externally visible identifiers. These could be used as the primary keys, or alternatively, the application can create identifiers that are not externally visible, and use them as the value for the primary key.
   As another example, the *section* table, which has the combination of (*course_id*, *section_id*, *semester*, *year*) as primary key, could instead have a section identifier that is unique across all sections as primary key, with the *course_id*, *section_id*, *semester*, *year* as non-primary key attributes. The difference would be that the relations that refer to *section*, namely *teaches* and *takes*, would have a single unique section id attribute as a foreign key referring to *section*, and would not need to store *course_id*, *section_id*, *semester*, and *year*.

Considerable emphasis is placed on the construction of tables from E-R diagrams. This serves to build intuition for the discussion of the relational model in the subsequent chapters. It also serves to ground abstract concepts of entities and relationships into the more concrete concepts of relations. Several other texts places this material along with the relational data model, rather than in the E-R model chapter. Our motivation for placing this material here is help students to appreciate how E-R data models get used in reality, while studying the E-R model rather than later on.

Exercises

7.14 Explain the distinctions among the terms primary key, candidate key, and superkey.

Answer: A *superkey* is a set of one or more attributes that, taken collectively, allows us to identify uniquely an entity in the entity set. A superkey may contain extraneous attributes. If K is a superkey, then so is any superset of K. A superkey for which no proper subset is also a superkey is called a *candidate key*. It is possible that several distinct sets of attributes could

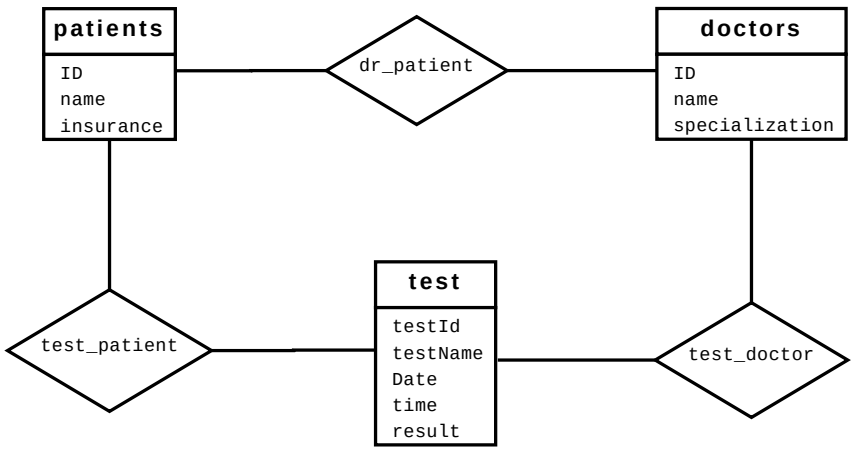


Figure 7.1 E-R diagram for a hospital.

serve as candidate keys. The *primary key* is one of the candidate keys that is chosen by the database designer as the principal means of identifying entities within an entity set.

7.15 Construct an E-R diagram for a hospital with a set of patients and a set of medical doctors. Associate with each patient a log of the various tests and examinations conducted.

Answer:

An E-R diagram for the hospital is shown in Figure 7.1. Although the diagram meets the specifications of the question, a real-world hospital would have many more requirements, such as tracking patient admissions and visits, including which doctor sees a patient on each visit, recording results of tests in a more structured manner, and so on.

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

Answer:

a. Car insurance tables:

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)

Note that a more realistic database design would include details of who was driving the car when an accident happened, and the damage amount for each car that participated in the accident.

b. 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 (ID, name, dept_name, tot_cred)
course(course_id, title, credits)
section(course_id, section_id, semester, year)
exam_marks(student_id, course_id, sec_id, semester, year, exam_id, marks)

c. Player Match tables:

match(match_id, date, stadium, opponent, own_score, opp_score)
player(player_id, name, age, season_score)
played(match_id,player_id, score)

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

Answer: See Figure 7.2. Note that we assume a player can play in only one team; if a player may switch teams, we would have to track for each match which team the player was in, which we could do by turning the relationship *played* into a ternary relationship.

7.18 Explain the difference between a weak and a strong entity set.

Answer: A strong entity set has a primary key. All tuples in the set are distinguishable by that key. A weak entity set has no primary key unless attributes of the strong entity set on which it depends are included. Tuples in a weak entity set are partitioned according to their relationship with tuples in a strong entity set. Tuples within each partition are distinguishable by a discriminator, which is a set of attributes.

7.19 We can convert any weak entity set to a strong entity set by simply adding appropriate attributes. Why, then, do we have weak entity sets?

Answer: We have weak entities for several reasons:

- We want to avoid the data duplication and consequent possible inconsistencies caused by duplicating the key of the strong entity.
- Weak entities reflect the logical structure of an entity being dependent on another entity.
- Weak entities can be deleted automatically when their strong entity is deleted.

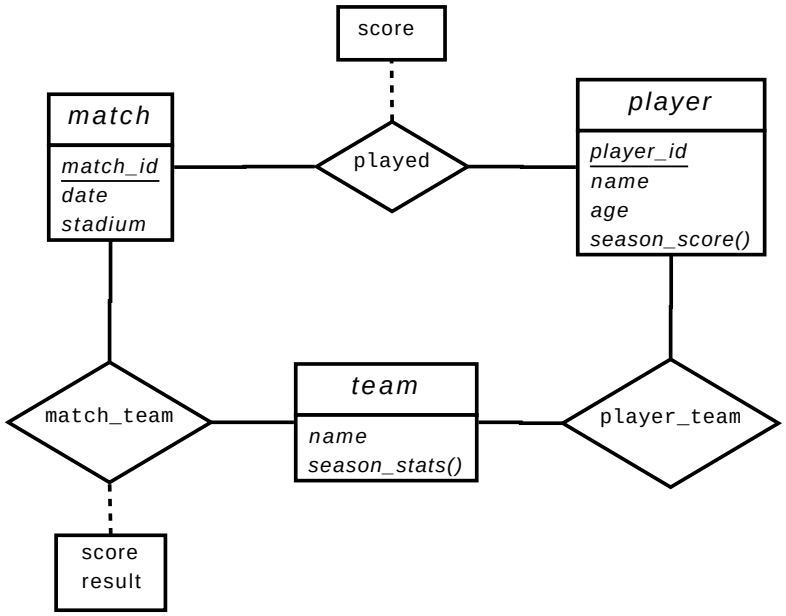


Figure 7.2 E-R diagram for all teams statistics.

- Weak entities can be stored physically with their strong entities.

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: Interpret the second part of the question as the bookstore adds videos, which may be in Blu-ray disk format or in downloadable format; the same video may be present in both formats.

a. Entity sets:

author(name, address, URL)
publisher(name, address, phone, URL)
book(ISBN, title, year, price)
customer(email, name, address, phone)
shopping_basket(basket_id)
warehouse(code, address, phone)

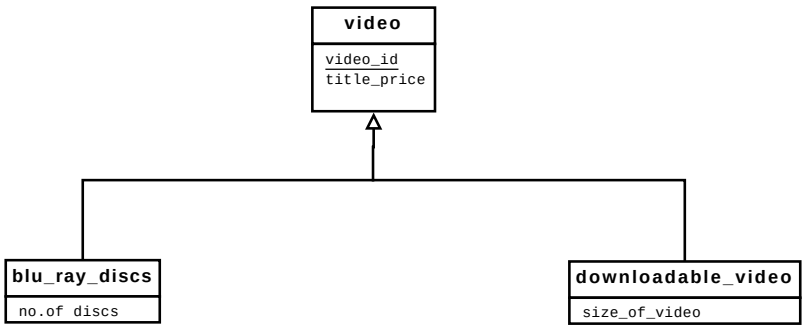


Figure 7.3 ER Diagram for Exercise 7.20 (b)

b. The ER-diagram portion related to videos is shown in Figure 7.3.

c. The E-R diagram shown in Figure 7.4 should be added to the E-R diagram of Figure 7.29. Entities that are shown already in Figure 7.29 are shown with only their names, omitting the attributes. The *contains* relationship in Figure 7.29 should be replaced by the version in Figure 7.4. All other parts of Figure 7.29 remain unchanged.

7.21 Design a database for an automobile company to provide to its dealers to assist them in maintaining customer records and dealer inventory and to assist sales staff in ordering cars.

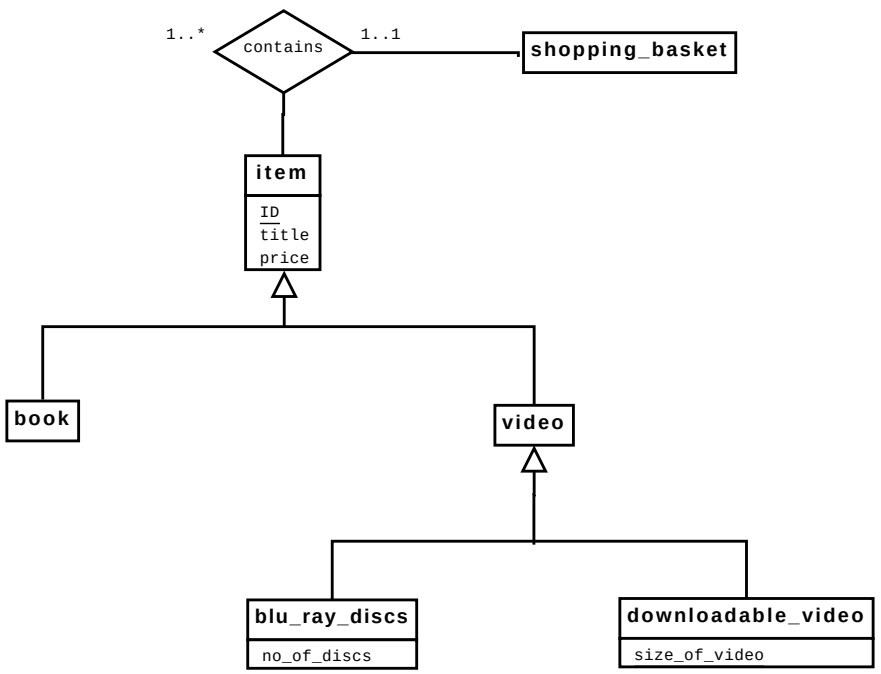


Figure 7.4 ER Diagram for Exercise 7.20 (c)

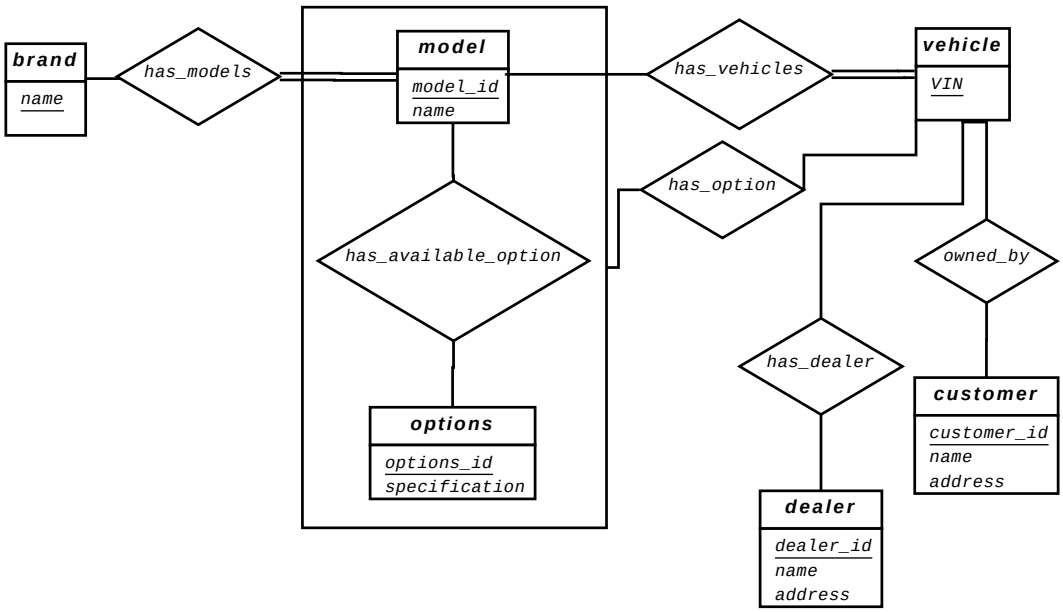


Figure 7.5 ER Diagram for Exercise 7.21

Each vehicle is identified by a vehicle identification number (VIN). Each individual vehicle is a particular model of a particular brand offered by the company (e.g., the XF is a model of the car brand Jaguar of Tata Motors). Each model can be offered with a variety of options, but an individual car may have only some (or none) of the available options. The database needs to store information about models, brands, and options, as well as information about individual dealers, customers, and cars.

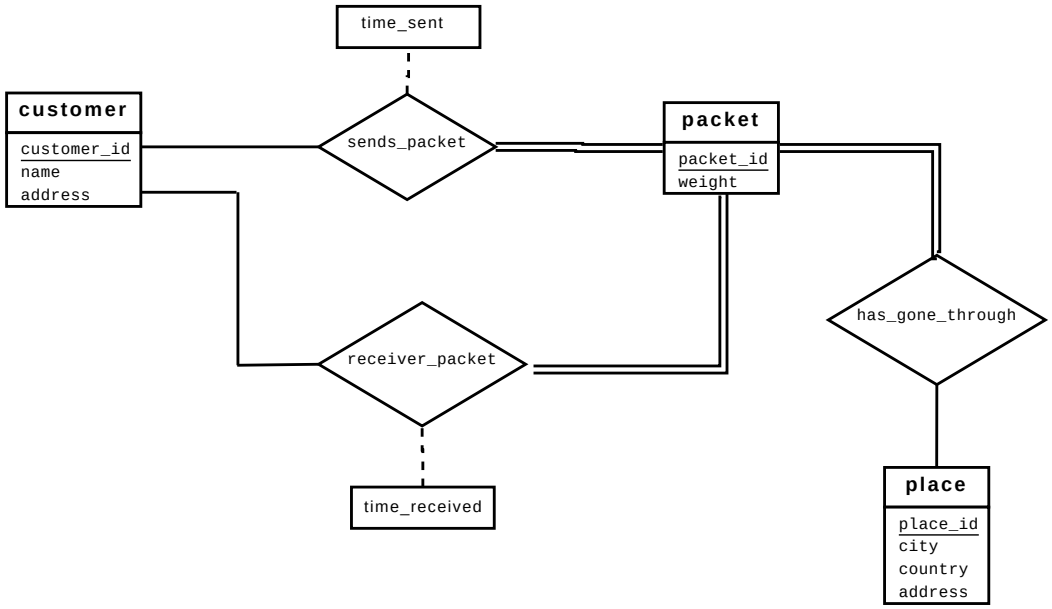
Your design should include an E-R diagram, a set of relational schemas, and a list of constraints, including primary-key and foreign-key constraints.

Answer:

The E-R diagram is shown in Figure 7.5. 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.

brand(name)
model(model_id,
name)
vehicle(VIN)
option(option_id,
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*
)



As an alternative:

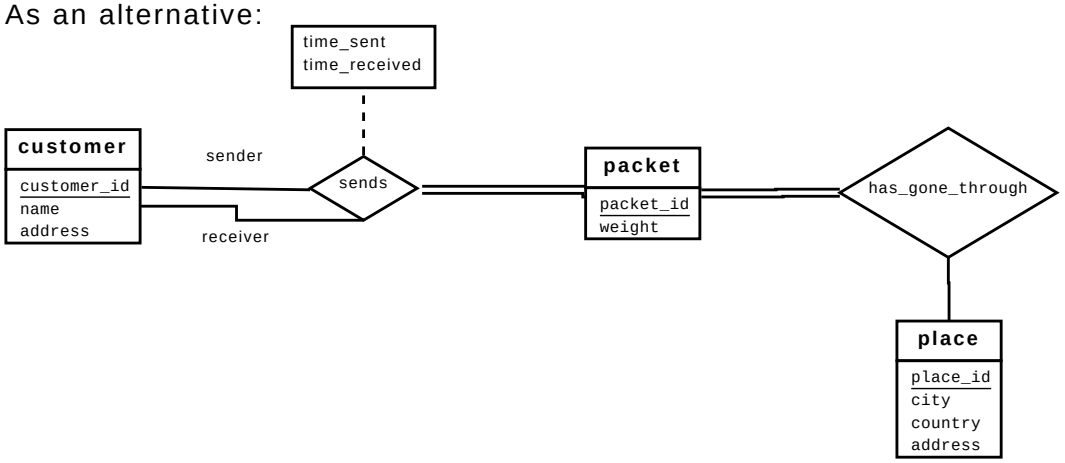


Figure 7.6 ER Diagram Alternatives for Exercise 7.22

7.22 Design a database for a world-wide package delivery company (e.g., DHL or FedEX). The database must be able to keep track of customers (who ship items) and customers (who receive items); some customers may do both. Each package must be identifiable and trackable, so the database must be able to store the location of the package and its history of locations. Locations include trucks, planes, airports, and warehouses.

Your design should include an E-R diagram, a set of relational schemas, and a list of constraints, including primary-key and foreign-key constraints.

Answer:

Two alternative E-R diagrams are shown in Figure 7.6.
The relational schema, including primary-key and foreign-key constraints, corresponding to the second alternative is shown below.

*customer(*customer_id*,*
name,
address)
*packet(*packet_id*,*
weight)
*place(*place_id*,*
city,
country,
address)
*sends(*sender_id*,*
receiver_id,
packet_id,
time_received,
time_sent
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*
)

7.23 Design a database for an airline. The database must keep track of customers and their reservations, flights and their status, seat assignments on individual flights, and the schedule and routing of future flights.

Your design should include an E-R diagram, a set of relational schemas, and a list of constraints, including primary-key and foreign-key constraints.

Answer:

The E-R diagram is shown in Figure 7.7. We assume that the schedule of a flight is fixed across time, although we allow specification of on which days a flight is scheduled. For a particular instance of a flight however we record actual times of departure and arrival. In reality, schedules change with time, so the schedule and routing should be for a particular flight for specified dates, or for a specified range of dates; we ignore this complexity.

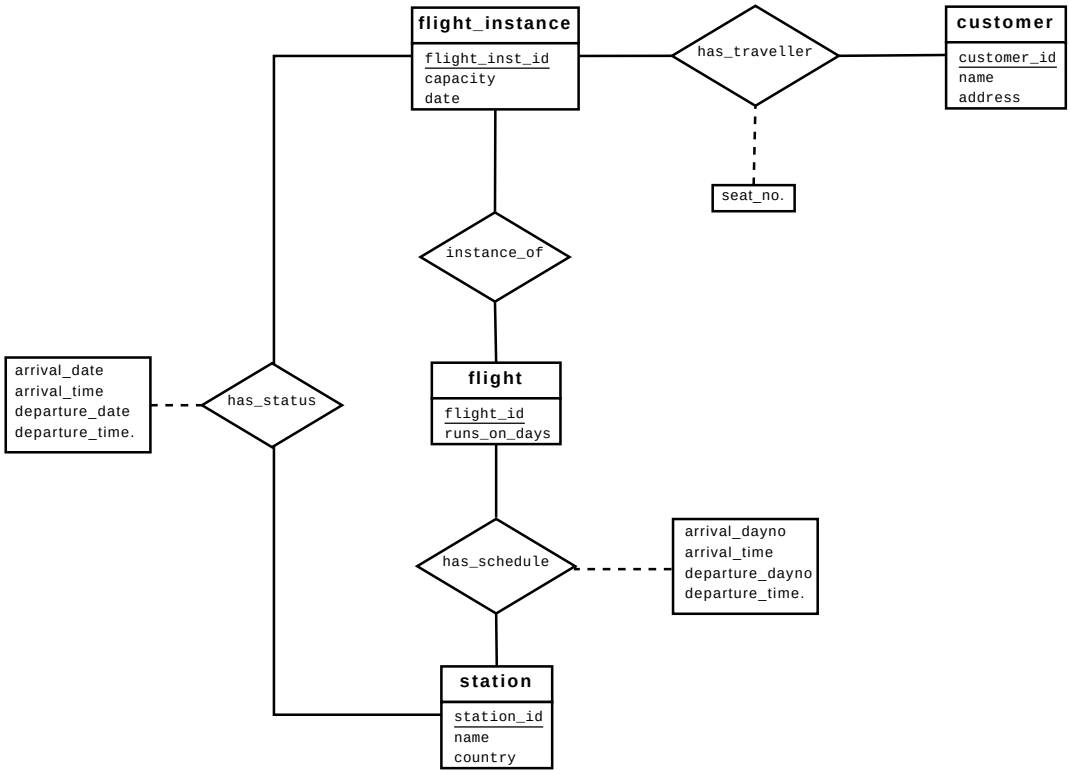


Figure 7.7 ER Diagram for Exercise 7.23

flight_instance(flight_inst_id, capacity, date)
customer(customer_id, name, address)
flight(flight_id, runs_on_days)
station(station_id, name, country)
has_traveller(
    flight_inst_id,
    customer_id,
    seat_number,
    foreign key *flight_inst_id* **references** *flight_instance,*
    foreign key *customer_id* **references** *customer*
)
instance_of(
    flight_inst_id,
    flight_id,
    foreign key *flight_inst_id* **references** *flight_instance,*
    foreign key *flight_id* **references** *flight*
)

has_schedule(
flight_id,
station_id,
order,
arrival_dayno,
arrival_time,
departure_dayno,
departure_time,
foreign key *flight_id* **references** *flight*,
foreign key *station_id* **references** *station*
)
has_status(
flight_inst_id,
station_id,
arrival_date,
arrival_time,
departure_date
departure_time
foreign key *flight_inst_id* **references** *flight_instance*,
foreign key *station_id* **references** *station*
)

7.24 In Section 7.7.3, we represented a ternary relationship (repeated in Figure 7.27a) using binary relationships, as shown in Figure 7.27b. Consider the alternative shown in Figure 7.27c. Discuss the relative merits of these two alternative representations of a ternary relationship by binary relationships.

Answer: In the model of Figure 7.27b, there can be instances where E, A, B, C, R_A, R_B and R_C cannot correspond to any instance of A, B, C and R.

The model of Figure 7.27c will not be able to represent all ternary relationships. Consider the ABC relationship set below.

A	B	C
1	2	3
4	2	7
4	8	3

If ABC is broken into three relationships sets AB, BC and AC, the three will imply that the relation (4, 2, 3) is a part of ABC.

7.25 Consider the relation schemas shown in Section 7.6, which were generated from the E-R diagram in Figure 7.15. For each schema, specify what foreign-key constraints, if any, should be created.

Answer: The foreign-key constraints are as specified below.

teaches(
    foreign key *ID* **references** *instructor,*
    foreign key *(course_id, sec_id, semester, year)* **references** *sec_course*
)

takes(
    foreign key *ID* **references** *student,*
    foreign key *(course_id, sec_id, semester, year)* **references** *sec_course*
)

prereq(
    foreign key *course_id* **references** *course,*
    foreign key *prereq_id* **references** *course*
)

advisor(
    foreign key *s_ID* **references** *student,*
    foreign key *i_id* **references** *instructor*
)

sec_course(
    foreign key *course_id* **references** *course,*
    foreign key *(sec_id, semester, year)* **references** *section*
)

sec_time_slot(
    foreign key *(course_id, sec_id, semester, year)* **references** *sec_course*
    foreign key *time_slot_id* **references** *time_slot*
)

sec_class(
    foreign key *(course_id, sec_id, semester, year)* **references** *sec_course*
    foreign key *(building, room_number)* **references** *classroom*
)

inst_dept(
    foreign key *ID* **references** *instructor*
    foreign key *dept_name* **references** *department*
)

stud_dept(
    foreign key *ID* **references** *student*
    foreign key *dept_name* **references** *department*
)

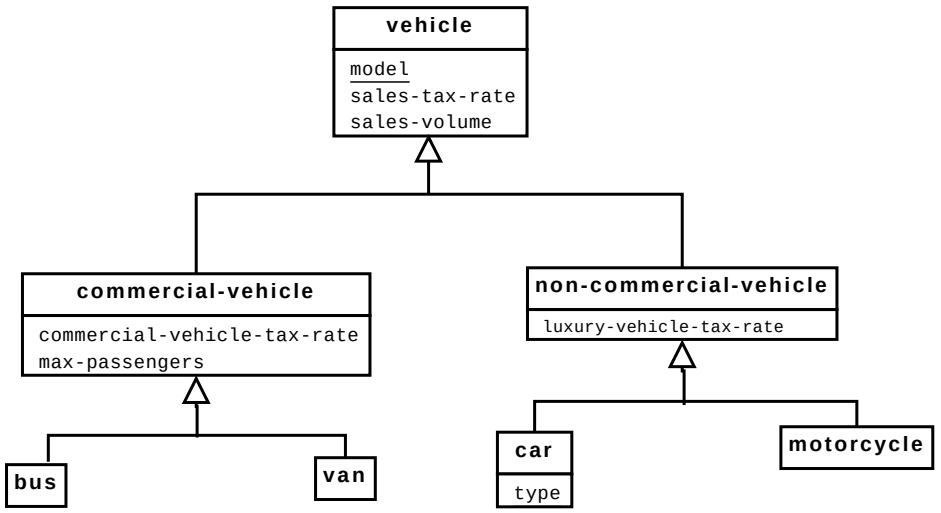


Figure 7.8 E-R diagram of motor-vehicle sales company.

course_dept(
foreign key *course_id* **references** *course*
foreign key *dept_name* **references** *department*
)

7.26 Design a generalization–specialization hierarchy for a motor vehicle sales company. The company sells motorcycles, passenger cars, vans, and buses. Justify your placement of attributes at each level of the hierarchy. Explain why they should not be placed at a higher or lower level.

Answer: Figure 7.8 gives one possible hierarchy; note that there could be many alternative solutions. The generalization–specialization hierarchy for the motor-vehicle company is given in the figure. *model*, *sales-tax-rate* and *sales-volume* are attributes necessary for all types of vehicles. Commercial vehicles attract commercial vehicle tax, and each kind of commercial vehicle has a passenger carrying capacity specified for it. Some kinds of non-commercial vehicles attract luxury vehicle tax. Cars alone can be of several types, such as sports-car, sedan, wagon etc., hence the attribute *type*.

7.27 Explain the distinction between condition-defined and user-defined constraints. Which of these constraints can the system check automatically? Explain your answer.

Answer: In a generalization–specialization hierarchy, it must be possible to decide which entities are members of which lower level entity sets. In a condition-defined design constraint, membership in the lower level entity-sets is evaluated on the basis of whether or not an entity satisfies an explicit condition or predicate.User-defined lower-level entity sets are not constrained by a membership condition; rather, entities are assigned to a given entity set by the database user.

Condition-defined constraints alone can be automatically handled by the system. Whenever any tuple is inserted into the database, its membership in the various lower level entity-sets can be automatically decided by evaluating the respective membership predicates. Similarly when a tuple is updated, its membership in the various entity sets can be re-evaluated automatically.

7.28 Explain the distinction between disjoint and overlapping constraints.

Answer: In a disjointness design constraint, an entity can belong to not more than one lower-level entity set. In overlapping generalizations, the same entity may belong to more than one lower-level entity sets. For example, in the employee-workteam example of the book, a manager may participate in more than one work-team.

7.29 Explain the distinction between total and partial constraints.

Answer: In a generalization–specialization hierarchy, a total constraint means that an entity belonging to the higher level entity set must belong to the lower level entity set. A partial constraint means that an entity belonging to the higher level entity set may or may not belong to the lower level entity set.