Database Software Design

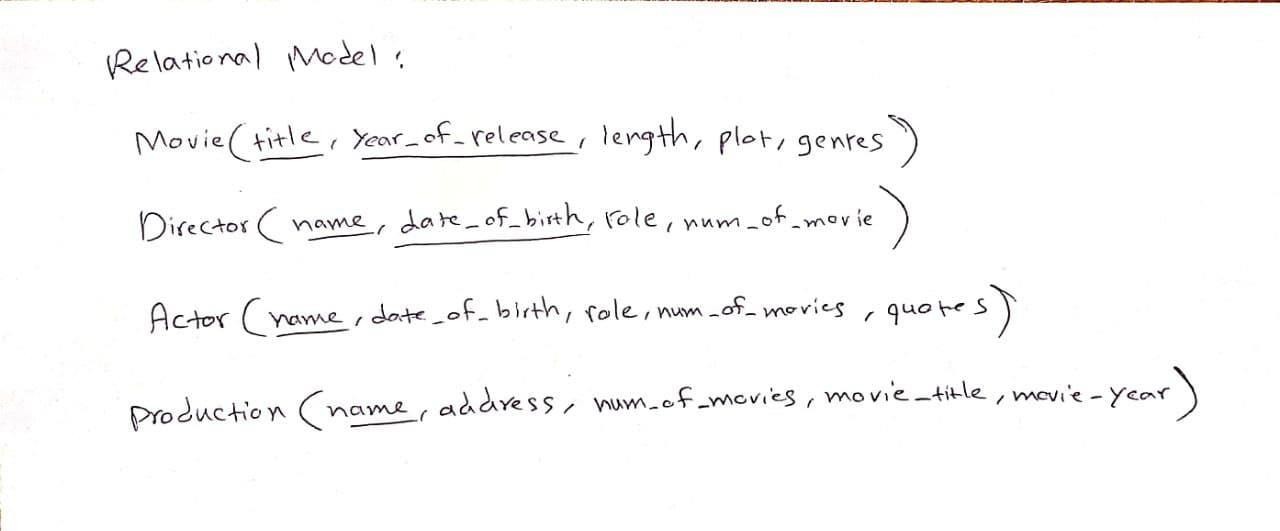
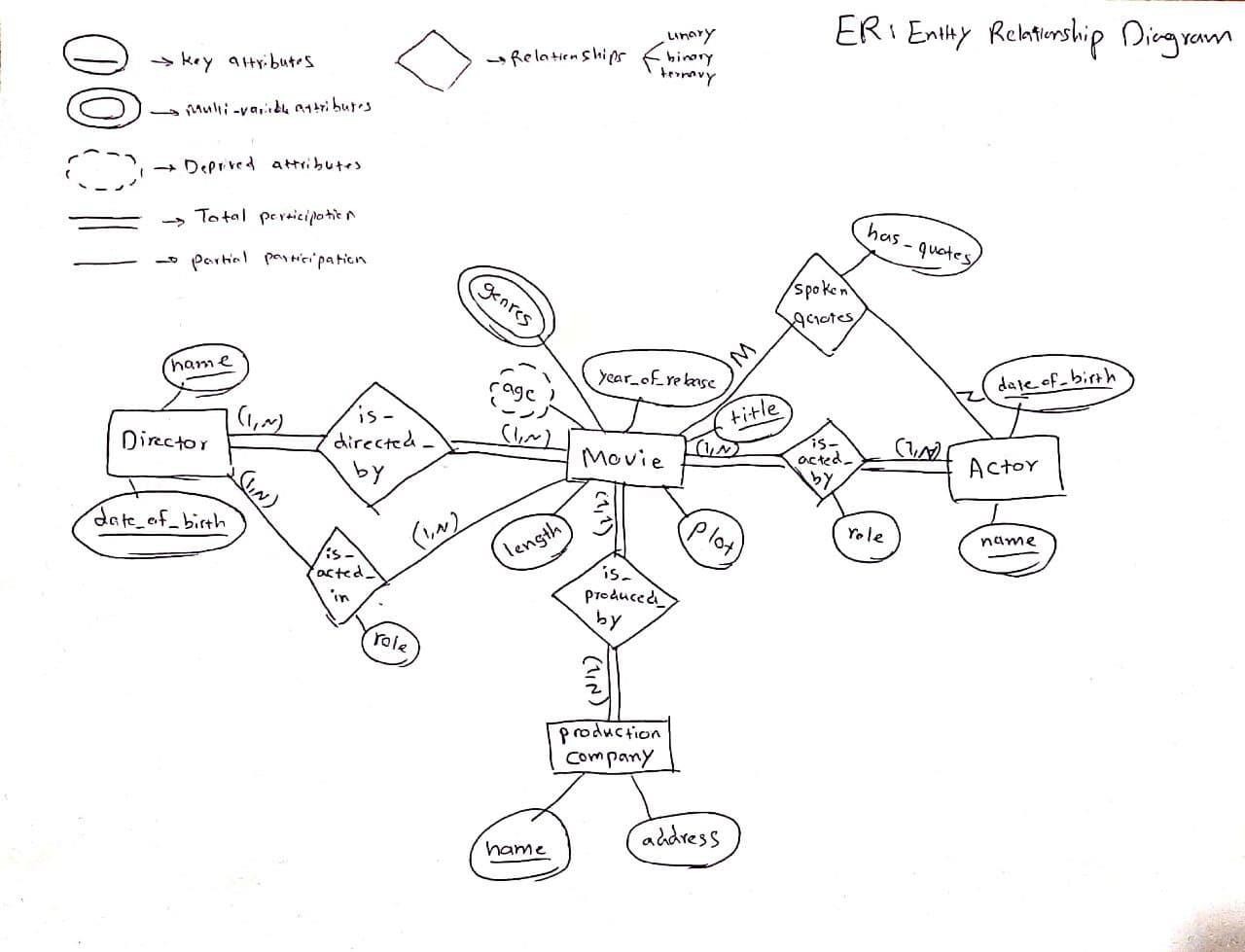
Modeling with UML

Consider a MOVIE database in which data is recorded about the movie industry. The data requirements are summarized as follows:

* Each movie is identified by title and year of release. Each movie has a length in minutes. Each has a production company, and each is classified under one or more genres (such as horror, action, drama, and so forth). Each movie has one or more directors and one or more actors appear in it. Each movie also has a plot outline. Finally, each movie has zero or more quotable quotes, each of which is spoken by a particular actor appearing in the movie.
* Actors are identified by name and date of birth and appear in one or more movies. Each actor has a role in the movie.
* Directors are also identified by name and date of birth and direct one or more movies. It is possible for a director to act in a movie (including one that he or she may also direct).
* Production companies are identified by name and each has an address. A production company produces one or more movies.

Design an Entity-Relationship diagram for the movie database and convert it to relational model.

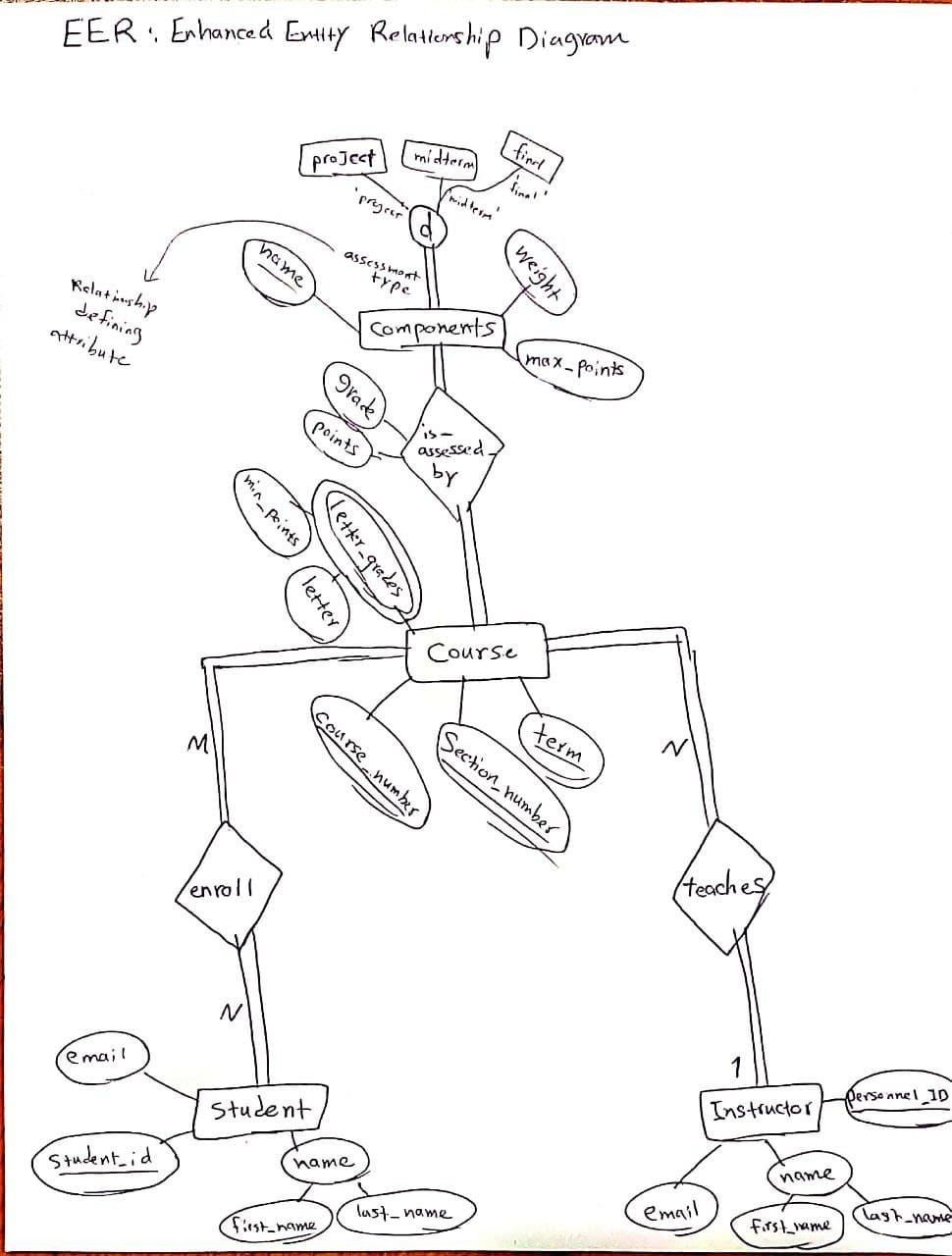
# Problem 2

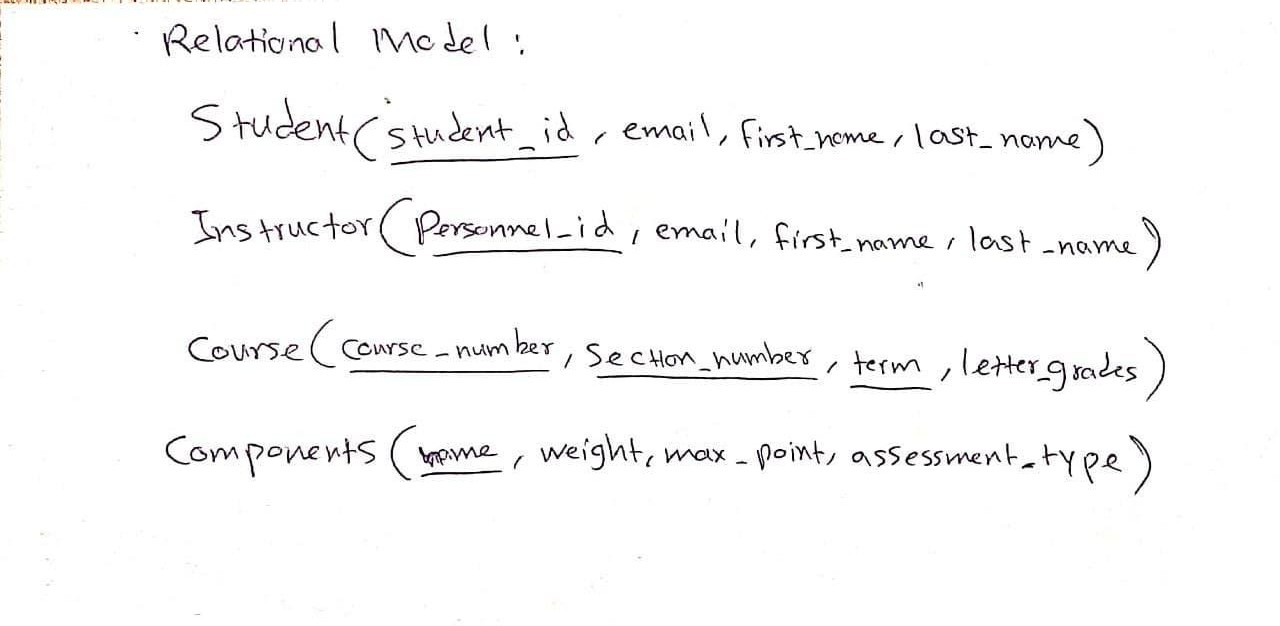


Consider a GRADE\_BOOK database in which instructors within academic department record points earned by individual students in their classes. The data requirements are summarized as follows:

* Each student is identified by a unique identifier, first and last name, and an e-mail address.
* Each instructor teaches certain courses each term. Each course is identified by a course number, a section number, and the term in which it is taught. For each course

he or she teaches, the instructor specifies the minimum number of points required in order to earn letter grades A, B, C, D, and F. For example, 90 points for an A, 80 points for a B, 70 points for a C, and so forth.

* Students are enrolled in each course taught by the instructor.
* Each course has a number of grading components (such as midterm exam, final exam, project, and so forth). Each grading component has a maximum number of points (such as 100 or 50) and a weight (such as 20% or 10%). The weights of all the grading components of a course usually total 100.
* Finally, the instructor records the points earned by each student in each of the grading components in each of the courses. For example, student 1234 earns 84 points for the midterm exam grading component of the section 2 course CSc2310 in the fall term of 2009. The midterm exam grading component may have been defined to have a maximum of 100 points and a weight of 20% of the course grade.

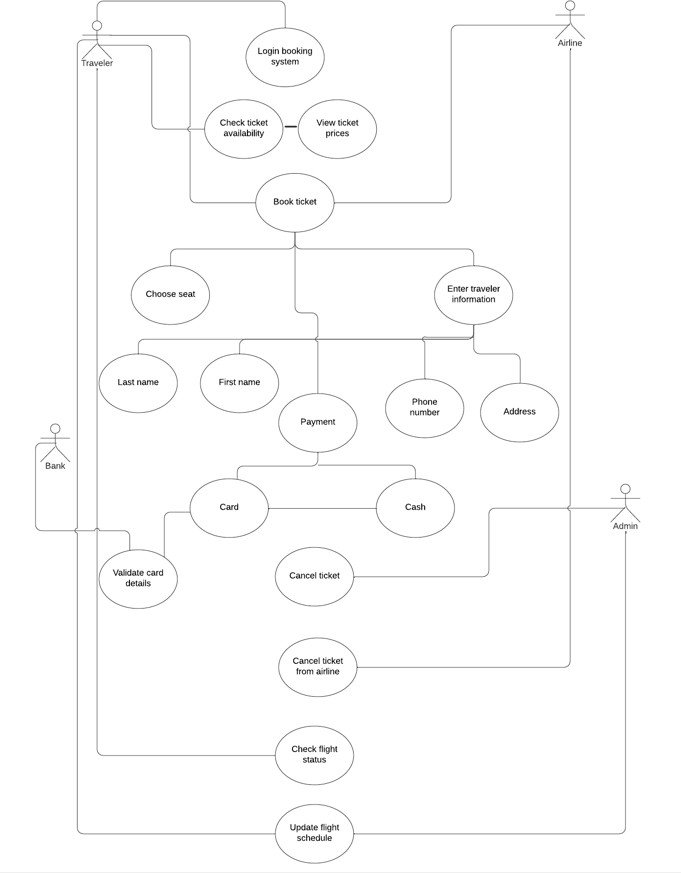
******

The airline AirForm proposes us to model in UML a reduced version of its information system of reservation of flight tickets. Flights are planned in advance and assigned to an airplane, an airport of departure, an airport of arrival, a departure date and an arrival date. Each airplane has a capacity in maximum number of passengers. Tickets are issued for each flight when planning, there is no overbooking.

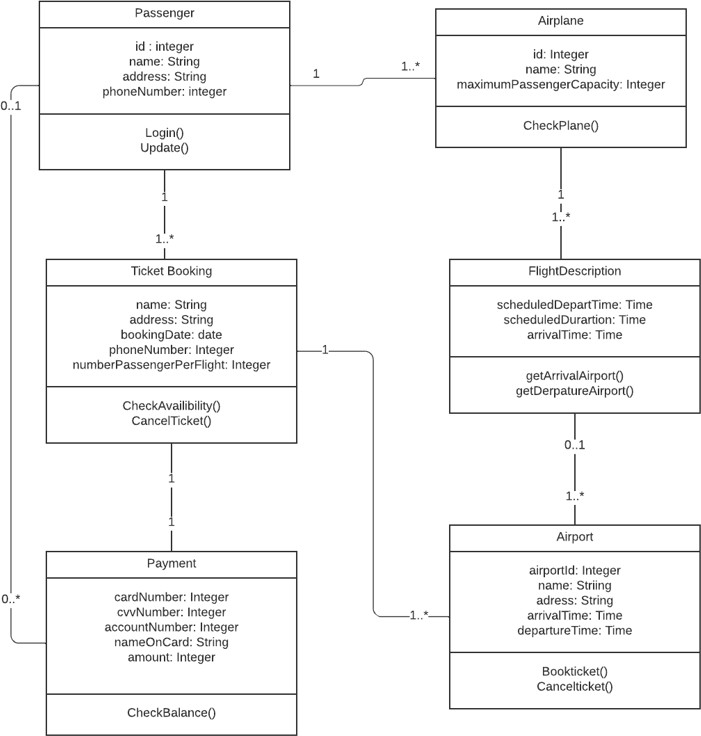
Users buy the tickets. This purchase results in a reservation (via ticket) for the flight in question. We keep the last names, first names, addresses and phones of the users who made a reservation, as well as the booking date and the ticket price. Upon check-in (departure), passengers confirm their tickets for the registered flight. We memorize this initial confirmation.

When the flight is over, the reservations associated with it are archived, and they are deleted when the flight is canceled.

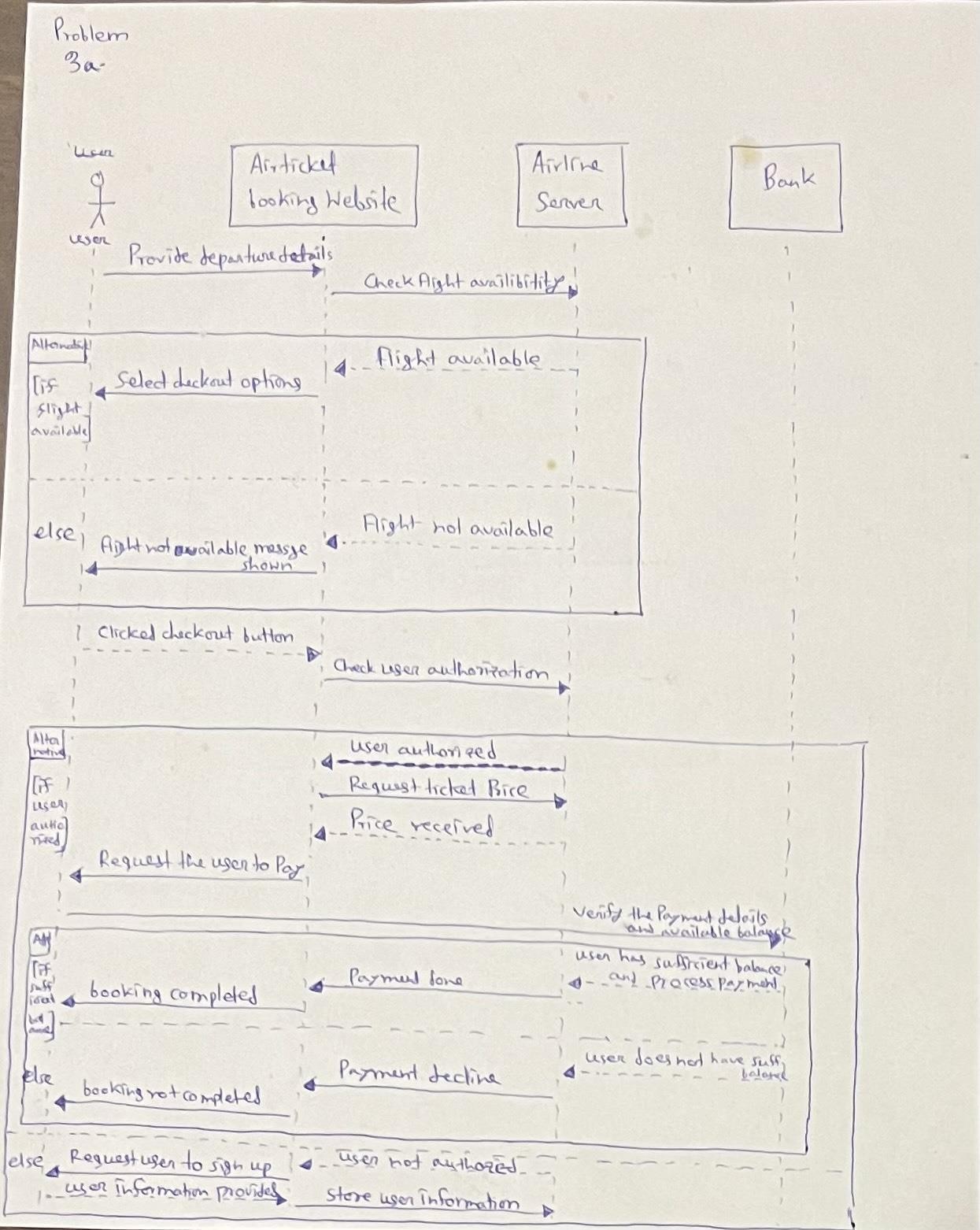
1. Write the high-level use case “Buy the tickets” initiated by a customer, and refine if it is possible this high-level use case.

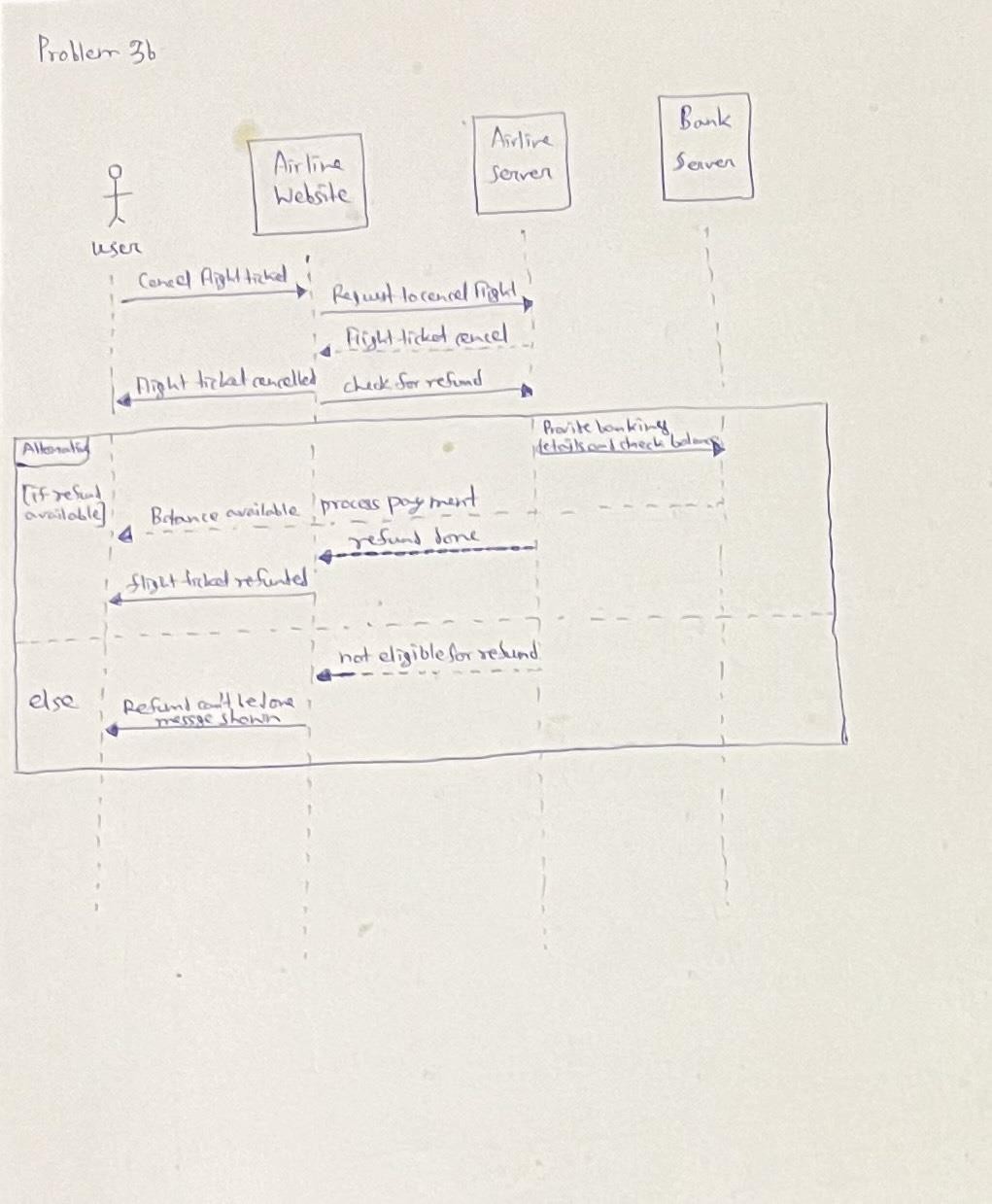


1. Propose a class diagram that models the AirForm system without representing other elements than those indicated in the statement. Please, specify the attributes for each class.



1. Model the following treatments with detailed sequence diagrams.
   1. Book a ticket on a flight.
   2. Cancel a flight.





# Programming with SQL Instructions

# Problem 1

You are going to use *PostgreSQL* to design a simple University database. You will create tables and implement some queries.

Create the tables described below. Name these tables TEACHER, COURSE, STUDENT, ENROLMENT, COURSE\_SCHEDULE.

TEACHER (t\_id : number, t\_name: text, t\_status:text , t\_dept: text) COURSE( c\_id : text, c\_name: text,c\_level :text)

STUDENT (s\_id : number, s\_name: text, s\_status: text) ENROLMENT (#c\_id : text, #s\_id : number ) COURSE\_SCHEDULE (#c\_id : text, #t\_id : number )

Primary Keys are underlined, and the foreign keys are preceded by the symbol #. Populate the tables you created by relying on the data provided in the file *populate.txt*. **Queries**

1. Find the name(s) of all teachers(s) who are from ECE department.
2. Find the name(s) of all student(s) enrolled in CS250
3. Find the student id(s) and names(s) of all students enrolled in CS348 and either in ECE264 or in CS503
4. Find the name of the teacher teaching MA525
5. Find the name(s) of all students enrolled in one or three courses
6. Find the name(s) of all students who are being taught by Prof. Christopher Clifton.
7. Name any undergraduate course(s) being taken by graduate student(s).
8. Name any undergraduate student(s) who is taking a course with Prof. Sheron Noel

# Problem 2

Suppose that you have the following university schema:

**create table** *classroom* (

*building* **varchar**(15), *room*\_*number* **varchar**(7), *capacity* **numeric**(4,0), **primary key** (*building*, *room*\_*number*)

);

**create table** *department*(

*dept*\_*name* **varchar**(20),

*building* **varchar**(15),

*budget* **numeric**(12,2) **check** (*budget* > 0),

**primary key** (*dept*\_*name*)

);

**create table** *course* (

*course*\_*id* **varchar**(8),

*title* **varchar**(50),

*dept*\_*name* **varchar**(20),

*credits* **numeric**(2,0) **check** (*credits* > 0),

**primary key** (*course*\_*id*),

**foreign key** (*dept*\_*name*) **references** *department*

# on delete set null

);

**create table** *instructor* (

*ID* **varchar**(5),

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

*dept*\_*name* **varchar**(20),

*salary* **numeric**(8,2) **check** (*salary* > 29000),

**primary key** (*ID*),

**foreign key** (*dept*\_*name*) **references** *department*

# on delete set null

);

**create table** *section* (

*course*\_*id* **varchar**(8),

*sec*\_*id* **varchar**(8),

*semester* **varchar**(6)

**check** (*semester in* ('Fall', 'Winter', 'Spring', 'Summer')),

*year* **numeric**(4,0) **check** (*year* > 1701 **and** *year* < 2100),

*building* **varchar**(15), *room*\_*number* **varchar**(7), *time*\_*slot*\_*id* **varchar**(4),

**primary key** (*course*\_*id*, *sec*\_*id*, *semester*, *year*),

**foreign key** (*course*\_*id*) references *course*

# on delete cascade,

**foreign key** (*building*, *room*\_*number*) **references** *classroom*

# on delete set null

);

**create table** *teaches* (

*ID* **varchar**(5),

*course*\_*id* **varchar**(8),

*sec*\_*id* **varchar**(8),

*semester* **varchar**(6),

*year* **numeric**(4,0),

**primary key** (*ID*, *course*\_*id*, *sec*\_*id*, *semester*, *year*),

**foreign key** (*course*\_*id*,*sec*\_*id*, *semester*, *year*) **references** *section*

# on delete cascade,

**foreign key** (*ID*) **references** *instructor*

# on delete cascade

);

**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

);

**create table** *takes* (

*ID* **varchar**(5),

*course*\_*id* **varchar**(8),

*sec*\_*id* **varchar**(8),

*semester* **varchar**(6),

*year* **numeric**(4,0),

*grade* **varchar**(2),

**primary key** (*ID*, *course*\_*id*, *sec*\_*id*, *semester*, *year*),

**foreign key** (*course*\_*id*,*sec*\_*id*, *semester*, *year*) **references** *section*

# on delete cascade,

**foreign key** (*ID*) **references** *student*

# on delete cascade

);

**create table** *advisor* (

*s\_ID* **varchar**(5),

*i\_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

);

**create table** *time*\_*slot* (

*time*\_*slot*\_*id* **varchar**(4),

*day* **varchar**(1),

*start*\_*hr* **numeric**(2) **check** (*start*\_*hr* >= 0 **and** *start*\_*hr* < 24), *start*\_*min* **numeric**(2) **check** (*start*\_*min* >= 0 **and** *start*\_*min* < 60), *end\_hr* **numeric**(2) **check** (*end*\_*hr* >= 0 **and** *end*\_*hr* < 24), *end\_min* **numeric**(2) **check** (*end*\_*min* >= 0 **and** *end*\_*min* < 60),

**primary key** (*time*\_*slot*\_*id*, *day*, *start*\_*hr*, *start*\_*min*)

);

**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*

);

After you create the previous tables in *pgAdmin* by importing the file *createTables.sql*, write the following queries in *SQL*, using the university schema. We suggest you to populate your tables by importing the file *PopulateData.sql* in *pgAdmin*.

1. Find the titles of courses in the Comp. Sci. department that have 3 credits.
2. Find the IDs of all students who were taught by an instructor named Einstein; make sure there are no duplicates in the result.
3. Find the highest salary of any instructor.
4. Find all instructors earning the highest salary (there may be more than one with the same salary).
5. Find the enrollment of each section that was offered in Autumn 2009.
6. Find the maximum enrollment, across all sections, in Autumn 2009.
7. Find the sections that had the maximum enrollment in Autumn 2009.

Suppose you are given a relation grade *points(grade, points)*, which provides a conversion from letter grades in the takes relation to numeric scores; for example an “A” grade could be specified to correspond to 4 points, an “A−” to 3.7 points, a “B+” to 3.3 points, a “B” to 3 points, and so on. The grade points earned by a student for a course offering (section) is defined as the number of credits for the course multiplied by the numeric points for the grade that the student received.

Given the above relation, and our university schema, write each of the following queries in SQL. You can assume for simplicity that no takes tuple has the null value for grade.

1. Find the total grade-points earned by the student with *ID* 12345, across all courses taken by the student.
2. Find the grade-point average (GPA) for the above student, that is, the total grade- points divided by the total credits for the associated courses.
3. Find the ID and the grade-point average of every student.
4. Increase the salary of each instructor in the Comp. Sci. department by 10%.
5. Delete all courses that have never been offered (that is, do not occur in the section relation).
6. Insert every student whose *tot\_cred* attribute is greater than 100 as an instructor in the same department, with a salary of $10,000.

# Problem 1

Write the following queries in SQL, using the university schema from the *assignment 2*.

1. Find the names of all students who have taken at least one Comp. Sci. course; make sure there are no duplicate names in the result.
2. Find the IDs and names of all students who have not taken any course offering before Spring 2009.
3. For each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor.
4. Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.
5. Create a new course “CS-001”, titled “Weekly Seminar”,with 0 credits.
6. Create a section of this course in Autumn 2009, with sec id of 1.
7. Enroll every student in the Comp. Sci. department in the above section.
8. Delete enrollments in the above section where the student’s name is Chavez.
9. Delete the course CS-001. What will happen if you run this delete statement without first deleting offerings (sections) of this course?
10. Delete all takes tuples corresponding to any section of any course with the word “database” as a part of the title; ignore case when matching the word with the title.
11. Suppose that we have a relation *marks(ID, score)* and we wish to assign grades to students based on the score as follows: grade F if *score < 40*, grade C if *40 ≤ score*

*< 60*, grade B if *60 ≤ score < 80*, and grade A if *80 ≤ score*. Write SQL queries to do the following:

* 1. Display the grade for each student, based on the marks relation.
  2. Find the number of students with each grade.

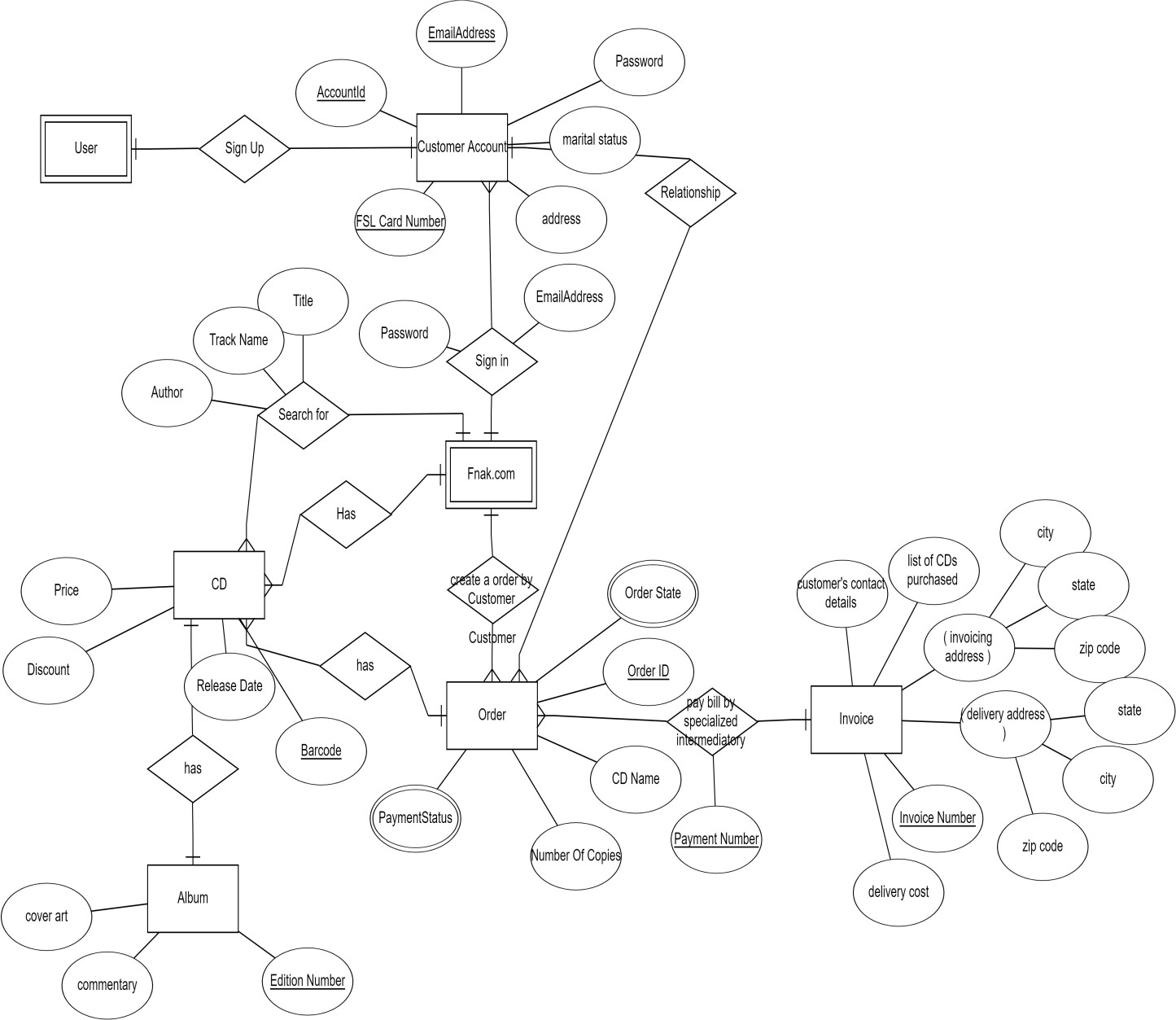
1. The SQL like operator is case sensitive, but the *lower()* function on strings can be used to perform case insensitive matching. To show how, write a query that finds departments whose names contain the string “sci” as a substring, regardless of the case.

**Modeling with ER**

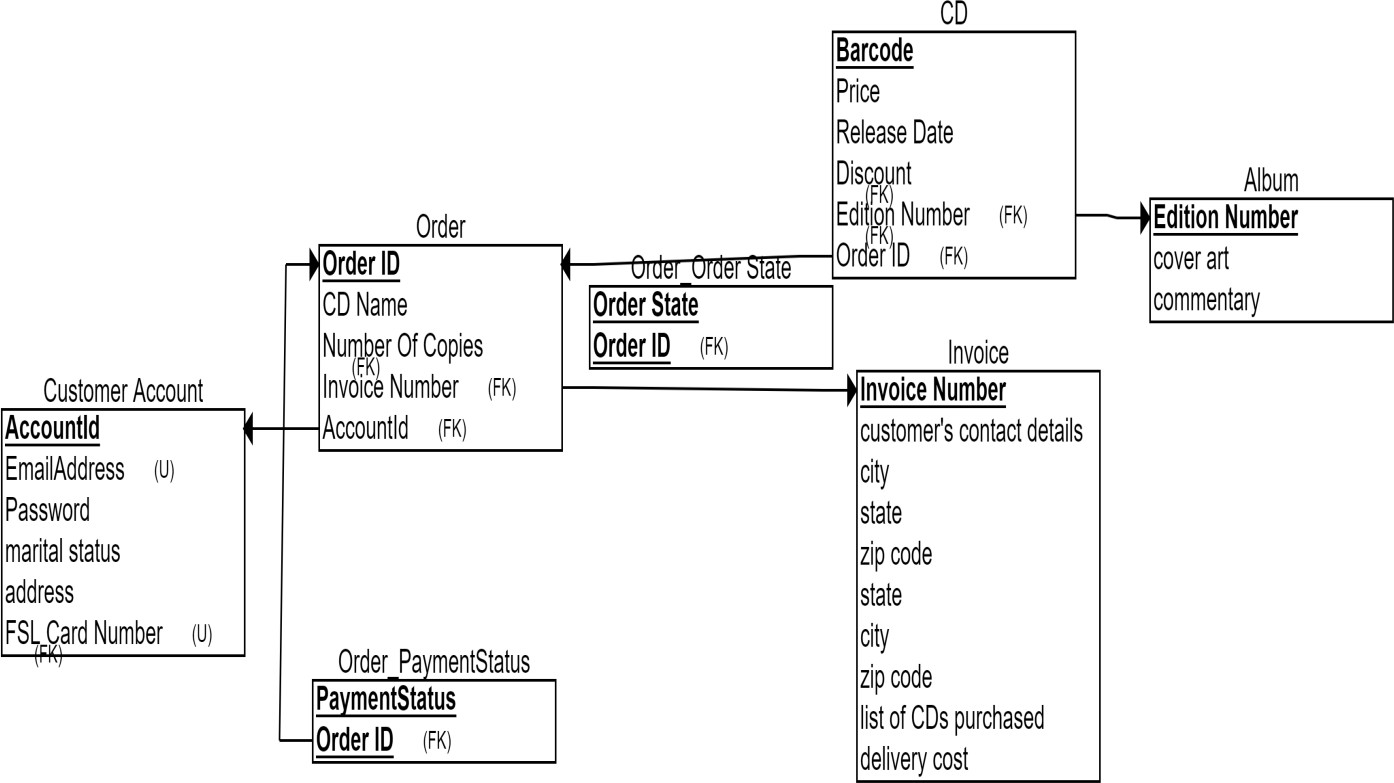
The manager of a CD sales chain ("Fnak") assigns you to create the information system for his Internet sales department ("Fnak.com"). To better understand the needs of this company, you questioned the director, and you were able to identify the following points:

* The only activity of Fnak.com is the sale of musical CDs to the French public on the Internet.
* The visitors to the online shopping site can only purchase after having opened a customer account to which they log in by entering their email address and a password of at least eight characters. When registering, they are asked for their marital status, their address (the one that will be used for invoicing the CDs purchased) and their Fnak store loyalty card number (if they have one).
* For each CD in the catalog, a visitor can view the album cover art and a commentary on that album. He can also know if it is a special edition of the album (a Collector's CD for example). The barcode number of a CD is different depending on the edition of the album. The price (expressed in euros) of a CD depends on the date from which it was released. In general, when it is released, its price is moderate, then a few weeks later, the price is increased. Finally, successful CDs are sold at a discount after a few months.
* During a visit to the site, a customer can create a new "basket" in which he puts the CD (s) he wants to buy as well as the number of copies he wants to buy. This basket is editable as long as payment for it is not validated and confirmed. Payment is made through a specialized intermediary who takes charge of deducting the cost of their order from customers, confirming to Fnak.com that the order has been paid for and issuing a unique payment number. . An invoice is then established, each invoice has a number (unique but the numbers do not necessarily follow), the customer's contact details, an invoicing address, a delivery address and the list of CDs purchased.
* There is a tool to search for a CD by its author, title or the name of a track on that CD. With each CD consulted, a list of other CDs is offered to the customer. This list is like "Internet users who bought this CD also bought:".
* Delivery costs are billed to customers: each order costs 3 euros plus 1 euro per CD ordered.
* An order can have several states after its validation: “awaiting replenishment” when one or more CDs are not in stock, “in process” when the order is in process, ”in delivery course ”when it is with the carrier (the customer can then consult the number of the corresponding postal package) then finally “delivered” when the carrier has confirmed that the package has been delivered.

*Design an Entity-Relationship diagram for the CD sales chain ("Fnak") and convert it to relational model.*



Relational schema



**Problem 2: Programming with SQL**

*Let the tables of a printing company that processes jobs for book publishers:*

*Publisher (****pubID****, pubName, street, city, postcode, telNo, creditCode)*

*BookJob (****jobID****, pubID, jobDate, description, jobType)*

*PurchaseOrder (****jobID****,* ***poID****, poDate)*

*POItem (****jobID****,* ***poID****,* ***itemID****, quantity)*

*Item (****itemID****, description, onHand, price)*

*where Publisher concerns details and* ***pubID*** *is the key.*

*BookJob concerns the printing jobs (books or part books) and* ***jobID*** *is the key.*

*PurchaseOrder A printing job requires the use of materials, such as paper and ink, which are assigned to a job via purchase orders. This table contains details of the purchase orders for each job and the key is* ***jobID/poID****. Each printing job may have several purchase orders assigned to it.*

*POItem Each purchase order (PO) may contain several PO items. This table contains details of the PO items and* ***jobID/poID/itemID*** *form the key.*

*and Item contains details of the materials which appear in POItem, and the key is* ***itemID****.*

Formulate the following queries using SQL:

-- 1. List all publishers in alphabetical order of name. select \* from Publisher order by pubName ASC;

-- 2. List all printing jobs for the publisher ‘Gold Press’.

select \* from BookJob bj join Publisher p on bj. pubID = p. pubID where p.pubName = 'Gold Press';

-- 3. List the names and phone numbers of all publisher who have a rush job (jobType = ‘R’).

select p.pubName, p.telNo from Publisher p join BookJob bj on p.pubID = bj.pubID where bj.jobType = 'R';

-- 4. List the dates of all the purchase orders for the publisher ‘Gold Press’.

select poDate from PurchaseOrder po join BookJob bj on po.jobID = bj.jobID join Publisher p on bj.pubID

= p.pubID where p.pubName = 'Gold Press'

-- 5. How many publisher fall into each credit code category?

select creditCode, count(p) numberOfPublisher from Publisher p group by creditCode;

-- 6. List all job type’s with at least three printing jobs.

select jobType from BookJob bj group by jobType having count(jobID)>=3;

-- 7. List the average price of all items. select AVG(price)::numeric(10,2) from Item;

-- 8. List all items with a price below the average price of an item. select \* from Item where (select AVG(price) mean from Item)> price ;

-- 9. Create a view of publisher details for all publisher who have a rush printing job, excluding their credit code.

CREATE VIEW Price\_View AS

SELECT pubID, pubName, street, city, postcode, telNo FROM Publisher p

right join BookJob bj on p.pubID = bj.pubID;

# Reliable Rentals Case Study

The requirements collection and analysis phase of the database design process has provided the following data requirements for a company called Reliable Rentals, which rents out vehicles (cars and vans). The Company has various outlets (garage/offices) throughout Glasgow. Each outlet has a number, address, phone number, fax number, and a manager who supervises the operation of the garage and offices at each site.

Each site is allocated a stock of vehicles for hire, however, individual vehicles may be moved between outlets, as required. Only the current location for each vehicle is stored. The registration number uniquely identifies each vehicle for hire and is used when hiring a vehicle to a client.

Clients may hire vehicles for various periods of time (minimum 1 day to maximum 1 year). Each individual hire agreement between a client and the Company is uniquely identified using a hire number. Information stored on the vehicles for hire include: the vehicle registration number, model, make, engine size, capacity, current mileage, date MOT due, daily hire rate, and the current location (outlet) of each vehicle.

The data stored on a hire agreement includes the hire number, the client’s number, name, address and phone number, date the client started the hire period, date the client wishes to terminate the hire period, the vehicle registration number, model and make, the mileage before and after the hire period. After each hire a member of staff checks the vehicle and notes any fault(s). Fault report information on each vehicle is stored, which records the name of the member of staff responsible for the check, date checked, whether fault(s) where found (yes or no), the vehicle registration number, model, make and the current mileage.

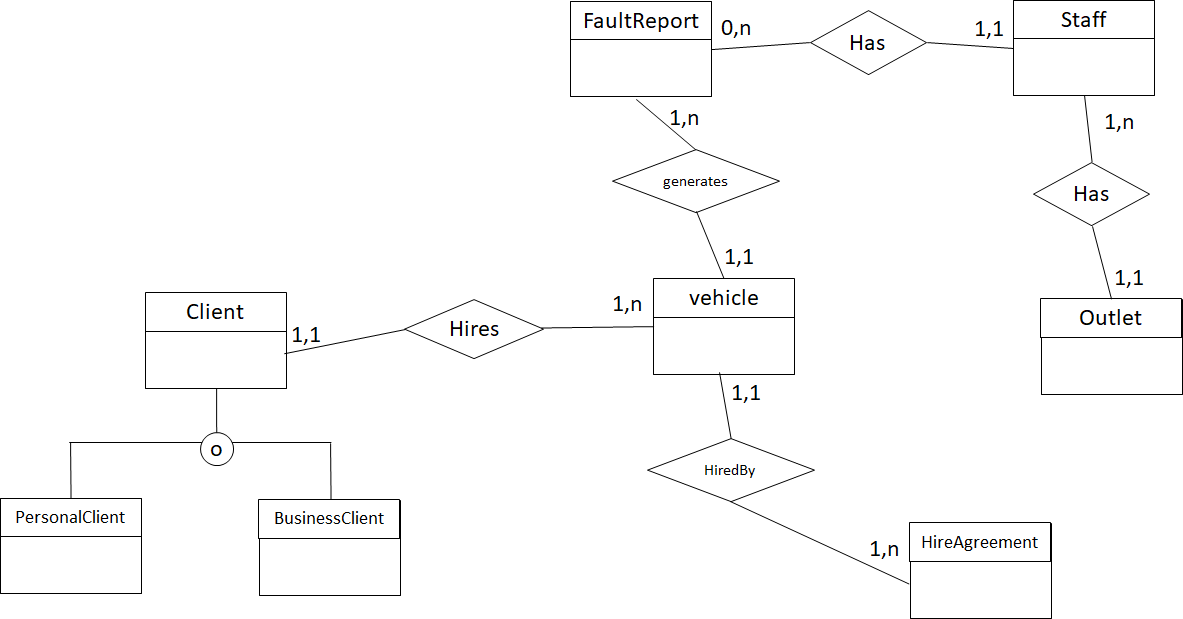
The Company has two types of clients: personal and business. The data stored on personal clients includes the client number, name (first and last name), home address, phone number, date of birth and driving licence number. The data stored on business clients

includes the client number, name of business, type of business, address, telephone and fax numbers. The client number uniquely identifies each client and the information stored relates to all clients who have hired in the past and those currently hiring a vehicle.

Information is stored on the staff based at various outlets including: staff number, name (first and last name), home address, home phone number, date of birth (DOB), sex, National Insurance Number (NIN), date joined the Company, job title and salary. Each staff member is associated with a single outlet but may be moved to an alternative outlet as required, although only the current location for each member of staff is stored.

*Create a conceptual schema for Reliable Rentals using the concepts of the Enhanced Entity–Relationship (EER) model. To simplify the diagram, only show entities, relationships and the primary key attributes. Specify the cardinality ratio and participation constraint of each relationship type. State any assumptions you make when creating the EER model (if necessary).*

***Solution***



# Programming with SQL

*The following tables form part of a database held in a Relational Database Management System:*

*Employee (****empID****, fName, lName, address, DOB, sex, position, deptNo) Department (****deptNo****, deptName, mgrEmpID)*

*Project (****projNo****, projName, deptNo) WorksOn (****empID****,* ***projNo****, hoursWorked)*

|  |  |  |
| --- | --- | --- |
| *where* | *Employee* | *contains employee details and* ***empID*** *is the key.* |
|  | *Department* | *contains department details and* ***deptNo*** *is the key. mgrEmpID* |
|  |  | *identifies the employee who is the manager of the department. There is only one manager for each department.* |
|  | *Project* | *contains details of the projects in each department and the key is*  ***projNo*** *(no two departments can run the same project).* |
| *and* | *WorksOn* | *contains details of the hours worked by employees on each project, and* ***empID/projNo*** *form the key.* |

1. *List all employees in alphabetical order of surname and within surname, first name.*

SELECT \*

FROM Employee

ORDER BY lName, fName;

1. *List all the details of employees who are female.*

SELECT \*

FROM Employee

WHERE sex = ‘F’;

1. *List the names and addresses of all employees who are Managers.*

SELECT fName, lName, address

FROM Employee

WHERE position = ‘Manager’;

Or

SELECT fName, lName, address FROM Employee e, Department d WHERE e.empID = d.mgrEmpID;

1. *Produce a list of the names and addresses of all employees who work for the ‘IT’ department.*

SELECT e.lName, e.address

FROM Employee e, Department d

WHERE e.deptNo = d.deptNo AND d.deptName = ‘IT’;

1. *Produce a complete list of all managers who are due to retire this year, in alphabetical order of surname.*

SELECT lName

FROM Employee e, Department d

WHERE e.empID = d.mgrEmpID AND

date\_part(‘year’,DOB) < date\_part(‘year’, DATE(‘2001-10-01’) – 65; (student does not need to know exact date functions – just general idea)

1. *Find out how many employees are managed by ‘James Adams’.*

SELECT COUNT(\*)

FROM Employees e1,e2, Department d

WHERE e1.lName = ‘Adams’ AND e1.fName = ‘James’ AND

e1.empID = d.mgrEmpID AND d.deptNo = e2.deptNo;

1. *Produce a report of the total hours worked by each employee, arranged in order of department number and within department, alphabetically by employee surname.*

SELECT e.empID, e.lName, e.fName, e.deptNo,

SUM(w.hoursWorked)

FROM Employee e, Project p, WorksOn w

WHERE e.deptNo = p.deptNo AND e.empID = w.empID

ORDER BY e.deptNo, e.lName;

1. *For each project on which more than two employees worked, list the project number, project name and the number of employees who work on that project.*

SELECT e.projNo, e.projName, COUNT(\*) FROM Project p, WorksOn w

WHERE p.projNo = w. projNo GROUP BY e.projNo, e.projName HAVING COUNT(\*) > 2;

1. *List the total number of employees in each department for those departments with more than 10 employees. Create an appropriate heading for the columns of the results table.*

SELECT deptNo AS departmentNumber, COUNT(empID) AS totalEmployees

FROM Employee

GROUP BY deptNo, empID

HAVING COUNT(empID) > 10;

1. *Create a view of employee details for all employees who work on project ‘MIS Development’, excluding department number.*

CREATE VIEW ED(eid, fName, lName, address, DOB)

AS SELECT empID, fName, lName, address, DOB

FROM Employee e, Project p

WHERE e.deptNo = p.deptNo AND p.projName = ‘MIS Development’;