Problem 1: (10 Points)

Indicate whether each of the following statement is true or false (**T**/**F**): (**1.25** points each).

F 1) It is known that every query that can be expressed in relational algebra can be expressed as a safe query in relational calculus; however, the converse may not be true.

F 2) The value of foreign key should be unique among all records existing in a destination relation instance.

F 3) In the relational calculus, a query describes the desired answer with specifying how the answer is to be computed.

F 4) A weak entity set, does not necessarily need to have a total participation in the identifying relationship set.

F 5) If a relation R is in 3NF, it is also in BCNF.

T 6) One advantage of normalization is that data modification operations usually execute faster.

T 7) Forcing a page to disk, ability to control pre-fetching and page replacement policy based on predictable access patterns are features supported by DBMS rather than OS.

F 8) Hash indexes are especially good at range queries.

Problem 2: (20 Points)

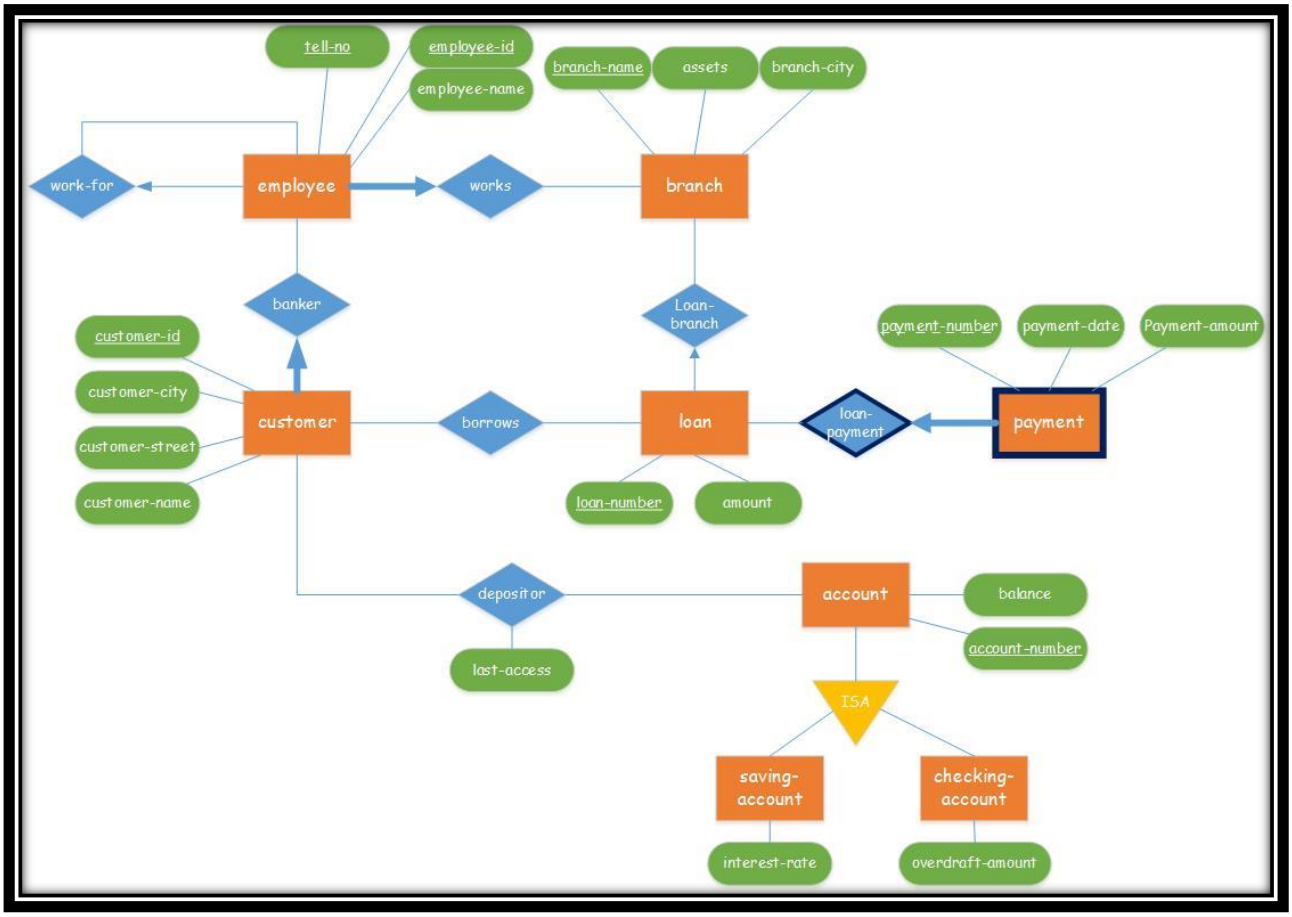
Data requirements for a banking enterprise described below:

- A bank is organized into branches. Each branch is located in a particular city and is identified by a unique name. We need to know total assets of each branch.
- Bank customers are identified by their *customer-id* values. The bank stores each customer's name, and the street and city where the customer lives. Customers may have accounts and can take out loans. Each customer have exactly one personal banker from employees.
- The bank offers two types of accounts, savings and checking accounts. Accounts can be held by more than one customer, and a customer can have more than one account. Each account is assigned a unique account number. The bank maintains a record of each account's balance, and the most recent date on which the account was accessed by each customer holding the account. In addition, each savings account has an interest rate, and overdrafts are recorded for each checking account.
- A loan originates at a particular branch and can be held by one or more customers. A loan is identified by a unique loan number. For each loan, the bank keeps track of the loan amount and the loan payments. Although a loan payment number does not uniquely identify a particular payment among those for all the bank's loans, a payment number does identify a particular payment for a specific loan. The date and amount are recorded for each payment.
- Bank employees are identified by their *employee-id* values. The bank administration stores the name and telephone number of each employee. Each employee may have a manager among employees. The bank need to know each employee work in which branch and each employee should work exactly in one branch.

a) Design an ER diagram that captures the operations for this bank. Use the notations taught in the class. Underline primary keys and show key constraints **(15 points)**.

b) Map your ER diagram to relational model. For each relation you need to mention the relation name, its attributes and underline the primary key fields. Also, mention the foreign keys. SQL descriptions are not required **(5 points)**.

a)



b)

branch (branch-name, branch-city, assets)

employee (employee-id, employee-name, tell-no, branch-name,
Foreign KEY (branch-name) REFERENCES branch)

works-for (employee-id, manager-id,
FOREIGN KEY (employee-id) REFERENCES employee (employee-id),
FOREIGN KEY (manager-id) REFERENCES employee (employee-id))

customer (customer-id, customer-name, customer-city, customer-street, banker-id,
FOREIGN KEY (banker-id) REFERENCES employee)

loan (loan-id, amount)

borrows (customer-id, loan-id,
FOREIGN KEY (customer-id) REFERENCES customer,
FOREIGN KEY (loan-id) REFERENCES loan)

loan-payment (payment-number, loan-id, payment-date, payment-amount,
FOREIGN KEY (loan-id) REFERENCES loan,
ON DELETE CASCADE)

account (account-number, balance)

checking-account (account-number, overdraft-protection,
FOREIGN KEY (account-number) REFERENCES account)

saving-acount (account-number, interest-rate,
FOREIGN KEY (account-number) REFERENCES account)

depositor (customer-id, account-number, last-access,
FOREIGN KEY (customer-id) REFERENCES customer,
FOREIGN KEY (account-number) REFERENCES account)

Problem 3: (10 Points) ☐☐

a) Consider the relational database described below, where the primary keys are underlined and foreign keys are **bolded** (**5** points total):

Sailors (sid, sname, rating, age)
Boats (bid, bname, color)
Reserves (**sid**, **bid**, day)

Shortly describe what the following queries do:

1. (**3** points)

Π_{sid} (Reserves) –

$\Pi_{sid}((\Pi_{sid}(\text{Reserves}) \times \sigma(\Pi_{bid}(\text{Boats}))) - \Pi_{sid, bid}(\text{Reserves}))$

Answer:
Finds sailors' sid who have reserved all boats.

2. (**2** points)
$\{<N> \mid \exists I, T, A, (<I, N, T, A> \in \text{Sailors} \wedge$
$\exists Br1, Br2, D1, D2(<I,Br1, D1> \in \text{Reserves}$
$\wedge <I, Br2, D2> \in \text{Reserves} \wedge Br1 \neq Br2))\}$

Answer:
Finds sailors' name who have reserved at least two different boats.

b) Explain what the below SQL query does and draw the table including records that the query will select (**5** points).
Consider the provided instance of Sailors:

sid	sname	rating	age
22	Dustin	7	45
29	Brutus	1	33
31	Lubber	8	55
32	Andy	8	25
58	Rusty	10	35
64	Horatio	7	35
71	Zorba	10	16
74	Horatio	9	35
85	Art	3	25
95	Bob	3	63
96	Frodo	3	25

```
SELECT S.rating, MIN (S.age) AS minage
FROM Sailors S
WHERE S.age >= 18
GROUP BY S.rating
HAVING COUNT(*) > 1
```

Finds the age of youngest sailor who is at least 18 years old for each rating level with at least two such sailors. (**3** points)

rating	minage
3	25
7	35
8	25

Problem 4: (25 Points) ☐☐

- This question, has 3 different parts and they are not necessarily correlated.
- RA = Relational Algebra, RC = Relational calculus

a) Consider the relational database schema described below, where the primary keys are underlined and foreign keys are **bolded** (**5** points total):

book (ISBN, title, author(s), publication)
customer (customer-name, customer-city, customer-street)
borrow (**customer-name**, **ISBN**)

1. Write a query to find customers' name, who has borrowed all books published by "McGraw-Hill". Write your queries in **RA** AND **SQL** AND **RC** (**5** points).

- **RA** (**1.5** points)

  $\Pi_{\text{cutomer-name, ISBN}}(\text{borrow}) / \Pi_{\text{ISBN}}(\sigma_{\text{publication = 'McGrawHill'}}(\text{book}))$

- **SQL** (**1.5** points) *cn = customer-name

```
SELECT BR.cn
FROM borrow BR
WHERE NOT EXIST ((SELECT BO.ISBN
                  FROM book BO)
                  EXCEPT
                  (SELECT BO2.ISBN
                   FROM borrow BR2
                   WHERE BR.cn = BR2.cn))
```

- **RC** (**1.5** points)

$$\{< C, CT, CS > \mid < C, CT, CS > \in customer \wedge$$
$$\forall IS, T, A, P\ (\ !(< IS, T, A, P >) \in book) \vee$$
$$(\exists CN, ISBN\ (< CN, ISBN > \in borrow \wedge C = CN \wedge IS = ISBN)))\}$$

b) Consider the relational database schema described below, where the primary keys are underlined and foreign keys are **bolded** (**12** points total):
employee (person-name, street, city)
works (**person-name**, **company-name**, salary)
company (company-name, city)
manages (**person-name**, **manager-name**)

1. Write a query to find the names and cities of residence of all employees who work for 'First Bank Corporation'. Write your queries in **SQL** AND **RC** (**3** points).

- **SQL** (**1.5** Points) * pn = person-name, cn = company-name
  SELECT E.pn, E.city
  FROM employee E JOIN works W on E.pn = W.pn
  WHERE W.cn = 'First Bank Corporation'
- **RC** (**1.5** Points)
  $\{< PN, PC > | < PN, PS, PC > \in employee \wedge$
  $\exists\ CN, S\ (< PN, CN, S > \in works) \wedge CN = \text{'First Bank Corporation'}\ )\}$

2. Write a query to find the names of all employees who live in the same city and on the same street as do their managers. Write queries in **RA** AND **SQL** (**3** points).

- **RA** (**1.5** points)
  ρ (e1, employee)
  ρ (m, managers)
  ρ (e2, employee)
  ρ (temp1, e1 $\bowtie$ m)
  ρ (temp2, σ m.person-name = e2.person-name (temp1 $\times$ e2))
  Π e1.person-name (σ e2.city = e1.city $\wedge$ e1.street = e2.street (temp2))

- **SQL** (**1.5** Points) * pn = person-name, cn = company-name
  WITH manemp as
  (SELECT E1.pn as e1pn, E1.city as e1ct, E1.street as e1st, M.pn as mpn
  FROM employee E1 JOIN manages M on E1.pn = M.pn)

```
SELECT ME.e1pn
FROM manemp ME, employee E2
WHERE ME.mpn = E2.pn AND
        ME.e1ct = E2.city AND
        ME.e1st = E2.street
```

3. Write a query to find the company with the most employees. Write a query in **SQL** (**2** points).

```
* cn = company-name, pn = person-name
WITH empcount as
(SELECT cn, count(pn) as pcount
FROM works W
GROUP BY cn)

SELECT ec.cn
FROM empcount ec
WHERE pcount = (SELECT MAX(pcount)
                FROM empcount)
```

4. Write a query to find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation. Write a query in **SQL** (**4** points)

```
* cn = company-name

SELECT W.cn
FROM works
GROUP BY W.cn
HAVING AVG(W.salary) > (SELECT AVG(W2.salary)
                        FROM works W2
                        WHERE W2.cn = 'First Bank Corporation')
```

c) Consider the relational database schema described below, where the primary keys are underlined and foreign keys are **bolded** (**8** points total):

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

Answer following questions using **SQL**.

a) Write a query to give all managers of First Bank Corporation a 10 percent raise unless the salary becomes more than $100,000 (**3** points).

* pn = person-name, mn=manager-name

```
UPDATE WORKS W
SET W.salary = (11 ÷ 10) × W.salary
WHERE W.cn = 'First Bank Corporation' and
W.salary <= 90909 and
W.pn IN (SELECT M.mn
         FROM manages M)
```

b) Add the following employee to the Database:
Person-name: 'Jones', street: 'Figueroa', City: 'LA'. Jones works at the 'Microsoft' and his annual salary is $90,000. Jones' manager is Bob. Assume that Microsoft already exists in the database (**2** points).

* pn = person-name, cn = company-name, mn = manager-name

```
INSERT INTO employee (pn, street, city) VALUES ('Jones', 'Figueroa', LA')
INSERT INTO works (pn, cn, salary) VALUES ('Jones', 'Microsoft', 90,000)
INSERT INTO manages (pn, mn) VALUES ('Jones', 'Bob')
```

c) Write an assertion to prevent the 'First Bank Corporation' and 'USC Bank' from having more than 1000 employees in aggregation (**3** points).

```
CREATE ASSERTION smallCompany
CHECK (
   (SELECT COUNT (pn) FROM works W WHERE W.cn = 'First
    Bank Co') +
   (SELECT COUNT(pn) FROM works W1 WHERE W1.cn = 'USC
    BANK') < 1000)
```

Problem 5: (20 Points) ☐☐

a) Does the FD set F = {A→BC, B→E, EC→F, FB→G} imply the FD A→G? Explain why? (**4** points).

Yes. A+ = {A, B, C, E, F, G} includes G. So the FDs imply A → G.

b) Compute Minimal Cover for the FD set provided below. Explain the procedure you use for computation (**6** points).

F = {ABD → EF, AB → G, B → F, C → J, CJ → I, G → E}

1. Put the FDs in Standard form

F: {ABD → E, ABD → F, AB → G, B → F, C → J, CJ → I, G → E}

2. Minimize the left side of each FD

For ABD → E, if remove A, then (BD)+ doesn't contain E, so can't remove A. similarly, B is not redundant as well. But (AB)+ ={A,B,G,E} contains E, so, we can remove D. ABD->E should be AB → E

For ABD → F, if remove A and D, then (B)+ contains F, so, A,D are redundant attributes.

ABD → F should be B → F

For AB → G, no redundant attribute.

For CJ → I, since C+={C, J, I} contains I, so J is redundant. CJ → I should be C → I

So, F={AB → E, B → F, AB → G, B → F, C → J, C → I, G → E }

3. Remove redundant FDs

First remove B → F, already has.

If remove AB → E, F = {AB → G, B → F, C → J, C → I, G → E}, and (AB)+ = {A,B,F,G,E} which contains E, so it is redundant

similarly, if remove AB → G,(AB)+={A,B,E}doesn't contain G,B+={B} doesn't contain F,C+ = {C,I} doesn't contain J. so, they are not redundant.

C → I, G → E are the same, they all can't be removed.

So the Minimal Cover F = {AB → G, B → F, C → J, C → I, G → E};

c) Given the relation R(A, B, C, D, E) with the FD set

F = {ABD → C, D → E, E → D, ABE → D} (**5** points Total):

1. Identify the candidate key(s) for R.(**2.5** Points)
2. Identify the best normal form that R satisfies (3NF or BCNF). Explain why? (**2.5** Points)

   The candidate key are ABD or ABE. C is not part of candidate key. There is no partial dependency and transitive dependency**.** So, it satisfies 3NF. But D and E are part of candidate key, so it doesn't satisfy BCNF.

d) Given relation R(X,Y,Z) with the FD set

F={Y → Z, Y → X, X → Y, X → Z} (**5** points total):

1. Identify the candidate key(s) for R. (**2.5** Points)
2. Identify the best normal form that R satisfies (3NF or BCNF). Explain why? (**2.5** Points)

   The candidate key is X and Y. Since X - Z and Y → Z, so Z is not transitive dependent on X or Y. Moreover, any FD contains candidate key, so it is BCNF.

Problem 6: (15 Points) ☐☐

Consider the following relations:

Professor (pid: integer, pname: varchar, sal: integer, age: integer, cid: integer)
Course (cid: integer, enrolled: integer, semester: varchar)

Salaries range from $10,000 to $100,000, ages vary from 20 to 80, each Course is taught by 4 professors on average, there are 3 semesters, and *enrolled* (including all sections of the corresponding course taught by all professors) vary from 20 to 150 students. You can assume uniform distributions of values.

For each of the following queries, which of the corresponding listed index choices would you pick to speed up the query? If your database system does not consider index-only plans, how would your answer change? Explain briefly.

a) Query: *Print pname, age, and sal of all Professors*. (**8** Points)

- ☐ Clustered hash index on <*pname*, *age*, *sal*>fields of Professor
- ☐ Unclustered hash index on <*pname*, *age*, *sal*>fields of Professor
- ☐ Unclustered hash index on <*pid*, *cid*>fields of Professor
- ☐ No index

We should create an unclustered hash index on <*pname*, *age*, *sal*>fields of Professor since then we could do an index only scan. If our system does not include index only plans then we shouldn't create an index for this query (d). Since this query requires us to access all the Professor records, an index won't help us any, and so should we access the records using a file scan.

b) Query: *Find the cids of courses that are opened on Fall semester and more than 50 students enrolled*. (**7** Points)

- ☐ Clustered hash index on the<*semester*, *enrolled*>field of Course
- ☐ Unclustered hash index on the<*semester*, *enrolled*>field of Course
- ☐ Clustered B+ tree index on the<*semester*, *enrolled*>fields of Course
- ☐ Unclustered B+ tree index on the<*semester*, *enrolled*>field of Course

We should create a clustered dense B+ tree index on <*semester*, *enrolled*>fields of Course, since the records would be ordered on these fields then. So when executing this query, the first record with *semester*= 'Fall' must be retrieved, and then the other records with *semester*= 'Fall' can be read in order of enrolled. Note that this plan, which is the best for this query, is not an index-only plan (must look up cids).