SAMPLE FINAL

A bank database contains information about:

1) Customers

2) Accounts

3) Employees

4) Branches

*Customers and Employees are uniquely identified by their **TCKimlikNo**. No employee is a customer. A customer is associated with exactly one branch. An employee is also associated with exactly one branch. Every customer has exactly one customer representative who is an employee of the associated branch. A bank branch is uniquely identified by its **branchNo**. Every account has an **accountNo.** An account belongs to exactly one branch. Different branches may use the same account number.*

The assumptions regarding the information recorded in the database are:

for each customer his/her TCKimlikNo (Social Security Number), name, branch, customer representative, phone number and address,

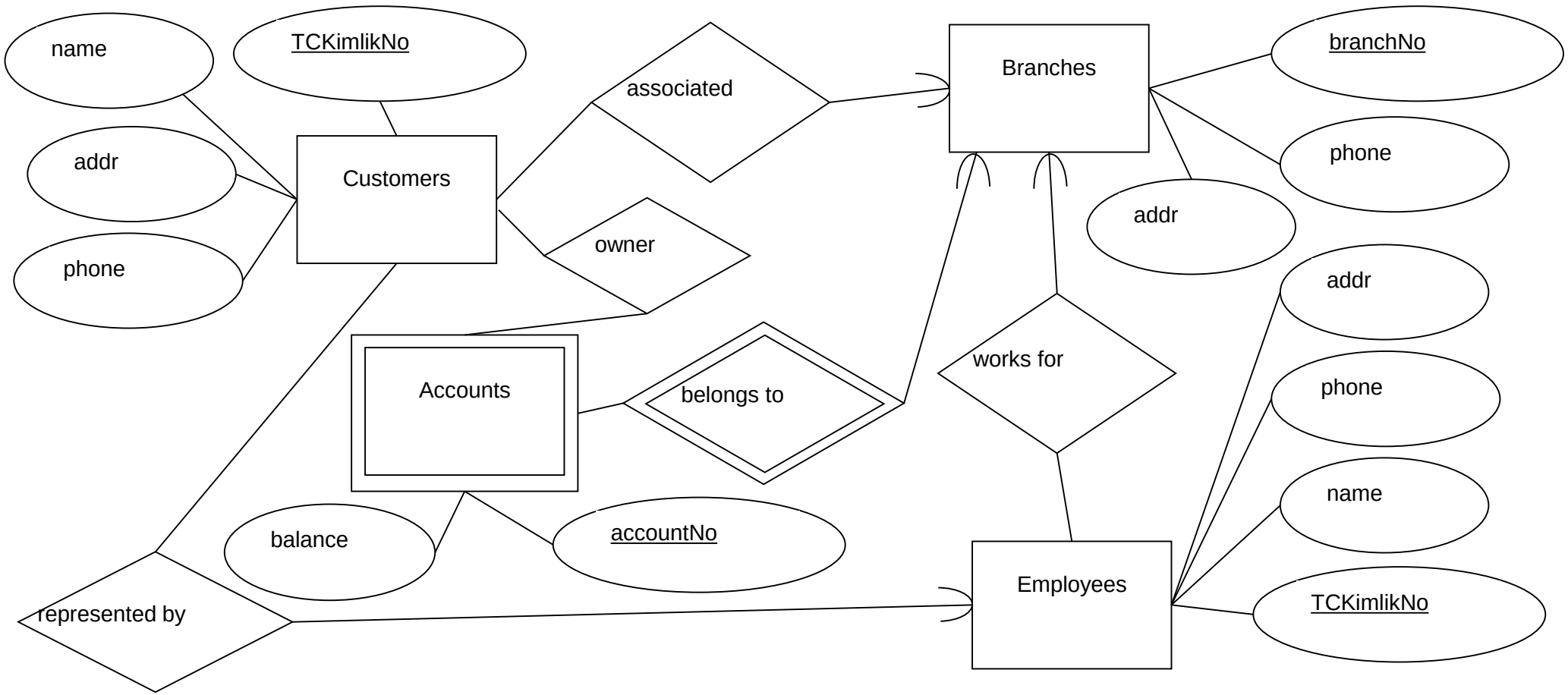
for each employee his/her TCKimlikNo (Social Security Number), name, branch, phone number and address,

for each branch, its branchNo, address and phone,

for each account, its branchNo, accountNo, owner TCKimlikNo and balance,

a customer may have more than one account, an account may be owned by more than one customer,

Draw the E/R Diagram for the registrar database system. Indicate keys, many-one relationships, weak entity sets, and other features of E/R diagrams.


name
TCKimlikNo
addr
Customers
phone
associated
owner
Branches
branchNo
phone
addr
Accounts
belongs to
works for
addr
phone
name
balance
accountNo
represented by
Employees
TCKimlikNo

Convert the E/R diagram obtained in 1-a) above to the relational data model using the straight E/R approach. Please do not forget to designate the keys for all the relations you obtain, and make any required optimizations along the way.

Solution:

1) Customers (TCKimlikNo, name, phone, addr, branchNo, repTCKimlikNo)

// - associated (TCKimlikNo, branchNo) is removed as it is combined with Customers

// - representedBy (custTCKimlikNo, empTCKimlikNo) is removed as it is combined with Customers

2) Employees (TCKimlikNo, name, phone, addr, branchNo)

// worksFor (TCKimlikNo, branchNo) is removed as it is combined with Employees

3) Branches (branchNo, phone, addr)

4) Accounts (branchNo, accountNo, balance)

5) owner (TCKimlikNo, accountNo)

Given the relational schema R (A, B, C, D, E) with MVDs A→→B, AB→→C and FDs A→D and AB→E

State whether the given relation R is in 4NF or not. Please explain why carefully.

Solution:
Examining the right hand sides of the given FDs, ABC is decided to be a part of every key. ABC+=ABCDE, and hence, is the only key. The left hand sides of all the MVDs along with the promoted FDs are readily seen to be not superkeys. As they are also non-trivial, they all form violations of 4NF.

R(A, B, C, D, E): MVDs: A→→B, AB→→C, FDs: A→D, AB→E

- All 4NF violations?

Keys(s) : ABC is always part of it. As ABC^{+}=ABCDE, ABC is the only key. Both the MVDs as well as the FDs promoted as MVDs violate 4NF.

- Decompose into 4NF as necessary?

Let us pick A→D promoted to start with. We obtain R1(A, D) FD: A→D and R2(A, B, C, E) FD: AB→E and MVDs: A→→B, AB→→C. R1 is in 4NF while R2 is not.

- Pick AB→E promoted to decompose R2:
- R21(A, B, E) FD: AB→E, MVD: A→→B and R22(A, B, C) MVDs: A→→B, AB→→C.
- Decomposing on A→→B in R21; R211(A, B) and R212(A, E) are obtained.
- AB→E has been lost or what. Using table test, check A→E? Apply A→D, A→→B, AB→E in this order. A→E holds and renders the original AB→E redundant.
- Moreover R211(A, B) MVD: A→→B and R212(A, E) FD: A→E are then both in 4NF.
- We are to last check R22(A, B, C) with MVDs: A→→B, AB→→C.
- Note that the set of new FDs: A→D, A→E and MVD: A→→B implies A→→C, which in turn renders AB→→C redundant (see the following slide).
- As A→→B is a violation, R22 is not in 4NF. R221(A, B) MVD: A→→B and R222(A, C) MVD: A→→C are obtained.

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	d_2=d_1	e_2=e_1
a	b_2	c_1	d_1	e_1
a	b_1	c_2	d_2=d_1	e_2=e_1

- As R221 = R211, ignore it.
- R222 is in 4NF.

Given R(A, B, C, D, E): MVDs: A→→B, AB→→C, FDs: A→D, A→E, prove that A→→C is valid, which in turn renders AB→→C redundant.

Proof: Use table test. Goal (a, b_1, c_2, d_1, e_1) and (a, b_2, c_1, d_2, e_2)

First use A→D, A→E to show that $d_2=d_1$ and $e_2=e_1$.

Then use A→→B to write two more tuples to obtain the goal

A	B	C	D	E
a	b_1	c_1	d_1	e_1
a	b_2	c_2	$d_2=d_1$	$e_2=e_1$
a	b_2	c_1	d_1	e_1
a	b_1	c_2	$d_2=d_1$	$e_2=e_1$

Given the database schema:

Product (maker, model, type)

PC (model, speed, ram, hd, price)

Laptop (model, speed, ram, hd, screen, price)

Printer (model, color, type, price)

use Embedded SQL in C to write the following SQL query: Ask the user for a price, and find the PCs whose price is closest the desired price. Print the maker, the model number, and the speed of the PC.

```
void closestMatchPC() {

EXEC SQL BEGIN DECLARE SECTION;

char manf, SQLSTATE[6]; int targetPrice, /* holds price given by user */

float tempSpeed, speedOfClosest; char tempModel[4], modelOfClosest[4]; int tempPrice, priceOfClosest; /* for tuple just read
from PC & closest price found so far */

EXEC SQL END DECLARE SECTION;


EXEC SQL DECLARE pcCursor CURSOR FOR SELECT model, price, speed FROM PC;

EXEC SQL OPEN pcCursor;

/* ask user for target price and read the answer into variable targetPrice */

/* Initially, the first PC is the closest to the target price. If PC is empty, we cannot answer the question, and so abort. */

EXEC SQL FETCH FROM pcCursor INTO :modelOfClosest, :priceOfClosest, :speedOfClosest; if(NOT_FOUND) /* print message
and exit */ ;


while(1) {

EXEC SQL FETCH pcCursor INTO :tempModel, :tempPrice, :tempSpeed; if(NOT_FOUND) break; if(/*tempPrice closer to
targetPrice than is priceOfClosest */) { modelOfClosest = tempModel; priceOfClosest = tempPrice; speedOfClosest =
tempSpeed; }

}

/* Now, modelOfClosest is the model whose price is closest to target. We must get its manufacturer with a single-row select
*/

EXEC SQL SELECT maker INTO :manf FROM Product WHERE model = :modelOfClosest; printf("manf=%s, model=%d,
speed=%d\n", manf, modelOfClosest, speedOfClosest);

EXEC SQL CLOSE CURSOR pcCursor;

}
```

Given the database schema:

Aircraft (aircraftType, capacity)
Passenger (pID, name, phone)
Flight (flightNo, aircraftType)
Reservation (pID, flightNo),

please state which trigger(s) is (are) needed to ensure that no reservations will cause an overbooking with regard to the capacity of the aircraft. You should also write one such trigger, named **capacity_control** that checks to see if a reservation causes overbooking (capacity of the aircraft exceeded) and rejects the reservation request in such a case. The trigger shall allow the reservation if the aircraft is not full.

```
CREATE TRIGGER capacity_control

AFTER INSERT ON Reservation

REFERENCING NEW ROW AS new_tuple

FOR EACH ROW

WHEN (

(SELECT COUNT (*) FROM Reservation WHERE  flightNo = new_tuple.flightNo)

>

(SELECT capacity FROM Flight f, Aircraft a WHERE new_tuple.flightNo = f.flightNo

        AND f.aircraftType = a.aircraftType )

)

DELETE FROM Reservation

WHERE flightNo = new_tuple.flightNo

AND pID = new_tuple.pID;
```

Consider the relation JoeSells(beer, price) and the following two transactions:

```
T1: BEGIN TRANSACTION
    S1: UPDATE JoeSells SET price=(3*price)
    WHERE beer='Bud';
    S2: UPDATE JoeSells SET price=(3*price)
    WHERE beer='Coors';
       COMMIT;
T2: BEGIN TRANSACTION
    S3: UPDATE JoeSells SET price=(2+price)
    WHERE beer='Coors';
    S4: UPDATE JoeSells SET price=(2+price)
    WHERE beer='Bud';
  COMMIT;
```

Assume the following committed table before either transaction executes:

JoeSells	beer	Price
	Bud	2
	Coors	3

You should additionally assume that 1) A transaction can see what it has written, even if it is not committed and 2) An update requires reading tuples as well as writing them.
Suppose the statements execute in this order:
S1, S3, S2, T1:COMMIT, S4, T2:COMMIT.
For the following cases, find the final prices of Bud and Coors.

T1 executes with READ UNCOMMITTED and T2 executes with READ COMMITTED. **(10 pts.)**

Stage	T1 sees	T2 sees	Committed	Uncommitted
Initial	B=2, C=3	B=2, C=3	B=2, C=3	None
After S1 (1 pt)	B=6, C=3	B=2, C=3	B=2, C=3	B=6
After S3 (1 pt)	B=6, C=5	B=2, C=5	B=2, C=3	B=6, C=5
After S2 (1 pt)	B=6, C=15	B=2, C=5	B=2, C=3	B=6, C=15
After T1 commits		B=6, C=15	B=6, C=15	None
After S4 (2 pts)		B=8, C=15	B=6, C=15	B=8
After T2 commits			B=8, C=15	None

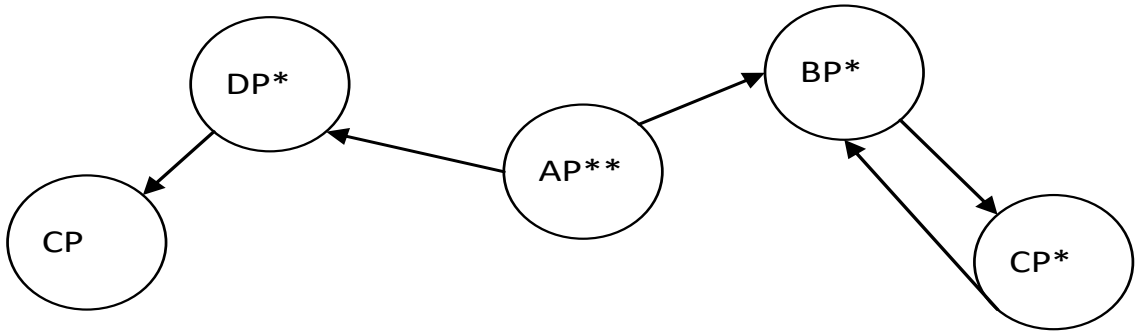
Bud: 8.
Coors: 15.

Both T1 and T2 execute with REPEATABLE READ.

Stage	T1 sees	T2 sees	Committed	Uncommitted
Initial	B=2, C=3	B=2, C=3	B=2, C=3	None
After S1 (1 pt)	B=6, C=3	B=2, C=3	B=2, C=3	B=6
After S3 (1 pt)	B=6, C=3	B=2, C=5	B=2, C=3	B=6, C=5
After S2 (1 pt)	B=6, C=9	B=2, C=5	B=2, C=3	B=6, C=9
After T1 commits		B=2, C=5	B=6, C=9	None
After S4 (2 pts)		B=4, C=5	B=6, C=9	B=4, C=5
After T2 commits			B=4, C=5	None

Bud: 4.
Coors: 5.

Given the following grant diagram



Write a possible sequence of SQL instructions that leads to the above diagram. Please, indicate for each instruction the authorization ID issuing it.

Step	By	Action
1	A	GRANT P TO B WITH GRANT OPTION
2	A	GRANT P TO D WITH GRANT OPTION
3	B	GRANT P TO C WITH GRANT OPTION
4	C	GRANT P TO B WITH GRANT OPTION
5	D	GRANT P TO C