CS 448/688
Assignment #1 Solution

Notes:
1. There are typically several possible correct solutions for each question (but there are many more possible incorrect solutions).
2. For simplicity, throughout this solution, the symbol ∞ indicates JOIN.

1.

Ex 4.5.2

a) π_{ename} ($\sigma_{aname='Boeing'}$ (Employees ∞ Certified ∞ Aircrafts))

b) { t[ename] | $e \in$ Employees $\wedge \exists c$ ($c \in$ Certified $\wedge$ c[eid] = e[eid] $\wedge$
$\exists a$ ($a \in$ Aircrafts $\wedge$ a[aid] = c[aid] $\wedge$ a[aname] = 'Boeing'))}

c)

Employees	*eid*	*ename*	*salary*
	_EID	P.ename	

Aircraft	*aid*	*aname*	*cruisingrange*
	_AID	'Boeing'	

Certified	*eid*	*aid*
	_EID	_AID

Ex 4.5.4

a) People have interpreted this in different ways. Some (including the book's authors) have interpreted it as requesting flights that can be flown by some pilot, in which case a solution is as follows:

π_{flno} ($\sigma_{cruisingrange \geq distance \wedge salary > 100,000}$ (Employees ∞ Certified ∞ Aircrafts ∞ Flights))

The proper answers use division, since you want all the high-paid pilots:

$\pi_{flno, eid, ename, salary}$ ($\sigma_{cruisingrange \geq distance}$ ((Employees ∞ Certified ∞ Aircrafts) $\times$ Flight))
$\div$ $\sigma_{salary > 100,000}$ (Employees)

(Note that since Flight has no common attributes with the other relations, natural join will provide the same result as cross-product.)

b) For the first form, we get the following:

{ f[flno] | $f \in$ Flights $\wedge \exists a, c, e$ ($a \in$ Aircraft $\wedge$ $c \in$ Certified $\wedge$ $e \in$ Employees $\wedge$
a[cruisingrange] $\geq$ f[distance] $\wedge$ e[salary] > 10000 $\wedge$ a[aid] = c[aid] $\wedge$
e[eid] = c[eid])}

For the second form, we get the following:

{f[flno] | $f \in$ Flights $\wedge \forall e$ ($e \in$ Employees $\wedge$ e[salary] > 100,000 $\wedge$
$\exists c$ ($c \in$ Certified $\wedge$ c[eid] = e[eid] $\wedge$
$\exists a$ ($a \in$ Aircraft $\wedge$ c[aid] = a[aid] $\wedge$ f[distance] $\leq$ a[cruisingrange])))) }

c)

Employees	*eid*	*ename*	*salary*
	_EID1		>100,000
	_EID2		>100,000

Certified	*eid*	*aid*
	_EID1	_AID

Aircraft	*aid*	*aname*	*cruisingrange*
	_AID		_CR

Flight	*flno*	*from*	*to*	*distance*	*departs*	*arrives*
	P.G.			_Dist		

Conditions
_CR ≥ _Dist COUNT._EID1 = COUNT._EID2

Ex 4.5.6

a) π_{eid} (Employee) – $\pi_{E1.eid}$ (ρ(E1,Employees) $\infty_{E1.salary < E2.salary}$ ρ(E2,Employees))

b) { $e1$[eid] | $e1 \in$ Employees $\wedge$ $\forall e2$ ($e2 \in$ Employees $\Rightarrow$ $e2$[salary] $\leq$ $e1$[salary])}
or
{$e1$[eid] | $e1 \in$ Employees $\wedge$ $\neg\exists e2$ ($e2 \in$ Employees $\wedge$ $e2$[salary]> $e1$[salary])}

c)

Employees	*eid*	*ename*	*salary*
	P.		_S1
			_S2

Conditions
S1 = *MAX*.S2

Ex 4.5.8

a) This query requires us to count the number of aircraft each pilot is certified for; however, the COUNT operation is not provided in the relational algebra. Thus this query is not expressible in relational algebra.

b) Same as part a), we don't have the required COUNT operation in tuple relational calculus, so this query is not expressible in tuple relational calculus.

c) This is possible to express in QBE, but it is tricky to compare the results of aggregated values. One method is to use an intermediate relation (as described in Section 6.9 in the text):

Certified	*eid*	*aid*
	G._EID	_AID

Counts	*eid*	*num*
I.	_EID	COUNT._AID

This defines a new relation and inserts corresponding tuples. Then we can write:

Counts	*eid*	*num*
	P.	_CNT
		_ ALLCNT

Conditions
CNT = MAX.ALLCNT

Ex 4.5.10

a) We require a SUM operation to do this query, but it is not provided in the relational algebra, so this query is not expressible in relational algebra.

b) Same as part a), we don't have the required SUM operation in tuple relational calculus, so this query is not expressible in tuple relational calculus.

c)

Employees	eid	ename	salary	
			_S	P.SUM._S

2. Ex 4.4.1

a) This query will produce an empty result or will be declared wrong by the compiler. This is because the projection of Parts on 'sid' does not make sense.

One possibility is to try a direct translation, but insert an impossible condition on the tuple variable ranging over Parts, as here:

$\{ s[\text{sname}] \mid s \in \text{Suppliers} \wedge \exists p\, (p \in \text{Parts} \wedge p[\text{color}] = \text{'red'} \wedge p \notin \text{Parts} \wedge$
$\quad \exists c\, (c \in \text{Catalog} \wedge c[\text{cost}] < 100 \wedge c[\text{sid}] = s[\text{sid}]))\}$

If we interpret the question as having a typo, and that the projection was meant to be on 'pid', we get:

$\{ s[\text{sname}] \mid s \in \text{Suppliers} \wedge \exists p\, (p \in \text{Parts} \wedge p[\text{color}] = \text{'red'} \wedge$
$\quad \exists c\, (c \in \text{Catalog} \wedge c[\text{cost}] < 100 \wedge c[\text{sid}] = s[\text{sid}] \wedge c[\text{sid}] = p[\text{pid}]))\}$

b) This cannot be expressed in QBE, as there is no 'sid' attribute in the Parts table. However, if we again treated it as a typo we would have gotten:

Suppliers	sid	sname	address
	_SID1	P._S	

Catalog	sid	pid	cost
	_SID1	_PID1	<100

Parts	pid	pname	colour
	_PID1		red

Ex 4.4.3

a) $\{ s[sname] \mid s \in Suppliers \wedge \exists p\, (p \in Parts \wedge p[color] = \text{'red'} \wedge$
$\exists c\, (c \in Catalog \wedge c[cost] < 100 \wedge c[pid] = p[pid] \wedge s[sid] = c[sid])) \wedge$
$\exists p2\, (p2 \in Parts \wedge p2[color] = \text{'green'} \wedge$
$\exists c2\, (c2 \in Catalog \wedge c2[cost] < 100 \wedge c2[pid] = p2[pid] \wedge$
$\exists s2\, (s2 \in Supplies \wedge s2[sid] = c2[sid] \wedge s[sname] = s2[sname]))) \}$

b)

Suppliers	sid	sname	address
	_SID1	P._S	
	_SID2	_S	

Catalog	sid	pid	cost
	_SID1	_PID1	<100
	_SID2	_PID2	<100

Parts	pid	pname	colour
	_PID1		red
	_PID2		green

Ex 4.4.5

a) $\{ s[sname] \mid s \in Suppliers \wedge \exists p\, (p \in Parts \wedge p[color] = \text{'red'} \wedge$
$\exists c\, (c \in Catalog \wedge c[cost] < 100 \wedge c[pid] = p[pid] \wedge s[sid] = c[sid])) \wedge$
$\exists p2\, (p2 \in Parts \wedge p2\, [color] = \text{'green'} \wedge$
$\exists c2\, (c2 \in Catalog \wedge c2[cost] < 100 \wedge c2[pid] = p2[pid] \wedge$
$\exists s2\, (s2 \in Supplies \wedge s2[sid] = c2[sid] \wedge s[sname] = s2[sname] \wedge$
$s[sid] = s2[sid]))) \}$

b)

Suppliers	sid	sname	address
	_SID	P.	

Catalog	sid	pid	cost
	_SID	_PID1	<100
	_SID	_PID2	<100

Parts	pid	pname	colour
	_PID1		red
	_PID2		green

3.

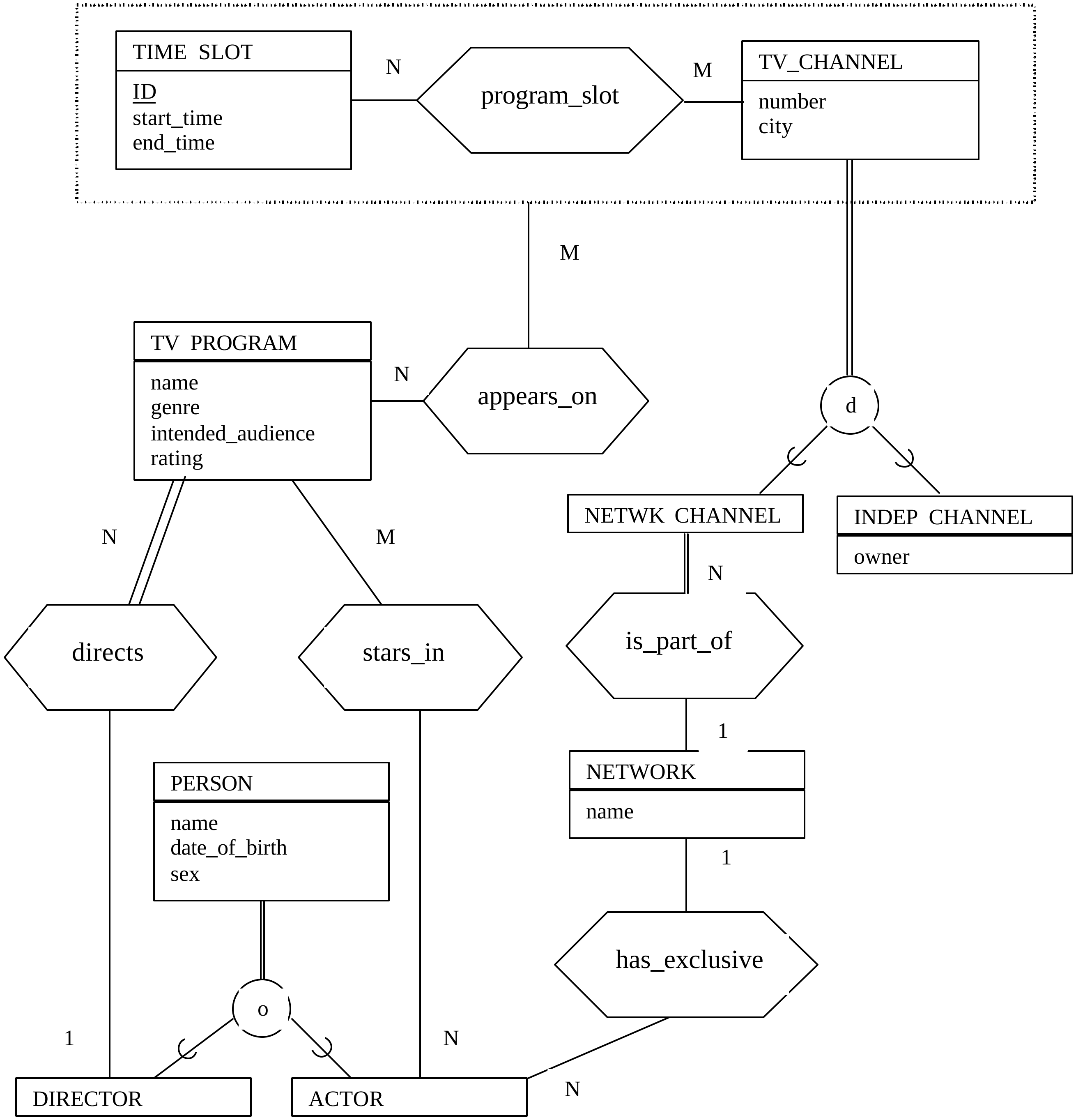
a) $\pi_{ENAME,RESP}\, (\sigma_{DUR>12}\, (EMP \infty WORKS))$

b) $\pi_{PNAME,RESP}\, (\sigma_{DUR>12 \vee BUDGET > 20000}\, (WORKS \infty PROJ))$

c) $\pi_{PNAME,ENAME}\, (EMP \infty PROJ \infty (\pi_{ENO,PNO}\, (WORKS) -$
$\pi_{ENO,PNO}\, (\rho(A,WORKS) \infty_{(A.RESP\ =\ B.RESP\ \wedge\ A.DUR<\ B.DUR)} \rho(B,WORKS))))$

4.

a) Note that we used a different notation to fit everything on one page.



Note: There is an additional unstated constraint that the time slots related to each TV channel must be pairwise non-overlapping.

b) Note: Primary keys are underlined and foreign keys are italic

Step 1 - Handling Entities:

TIME_SLOT(<u>ID</u>,start_time,end_time)
TV_CHANNEL(<u>number</u>, city)
TV_PROGRAM(<u>name</u>, genre, intended_audience, rating)
PERSON(<u>name</u>, date_of_birth,sex)
NETWORK(<u>name</u>)

Step 4 - 1:N Relationships

Modify relations to include foreign keys:

TV_PROGRAM(<u>name</u>, genre, intended_audience, rating, *director_name:* f.k. to DIRECTOR.name)
ACTOR(<u>name</u>, *exclusive_network*: f.k. to NETWORK.name)
TV_CHANNEL(<u>number</u>, city, type, owner, *network_name*: f.k. to NETWORK.name)

Add the constraint that network_name is not null iff type = “network”

Step 5: M:N Relationships

Add further tables:

STARS_IN(<u>*program_name*</u>: f.k. to TV_PROGRAM.name, <u>*actor_name*</u>: f.k. to ACTOR.name)
PROGRAM_SLOT(<u>*time_slot*</u>: f.k. to TIME_SLOT.ID, <u>*channel*</u>: f.k. to TV_CHANNEL.number)

Step 8: Specialization

When we use general specialization for Person we get:

ACTOR(<u>name</u>)
DIRECTOR(<u>name</u>)

Add the constraint that Actor.name and Director.name should be in Person.name

When we use disjoint specialization for the TV_Channel we get:

TV_CHANNEL(<u>number</u>, city, type, owner)

Add the constraint that type is either “network” or “independent”

Step 9: Aggregation

APPEARS_ON(*time_slot*: f.k. to TIME_SLOT.ID, *channel*: f.k. to TV_CHANNEL.number, *program_name*: f.k. to TV_PROGRAM.name)

Final set of relations:

TIME_SLOT (ID, start_time, end_time)
TV_CHANNEL(number, city, type, owner, *network_name*)
PROGRAM_SLOT (*time_slot*, *channel*)

TV_PROGRAM (name, genre, intended_audience, rating, *director_name*)
APPEARS_ON (*time_slot*, *channel*, *program_name*)

PERSON (name, date_of_birth, sex)
DIRECTOR (name)
ACTOR (name, *exclusive_network*)
STARS_IN (*program_name*, *actor_name*)

NETWORK (name)

As well as key constraints and foreign key constraints, we also have various domain constraints (i.e., whatever datatypes we would wish to associate with each attribute) plus inclusion dependencies between DIRECTOR.name and PERSON.name and between ACTOR.name and PERSON.name. We also have further constraint that TV_CHANNEL.network_name is not null iff TV_CHANNEL.type = “network”. (Note that because of this constraint we could eliminate the attribute TV_CHANNEL.type and use a test for NULL in TV_CHANNEL.network_name as the subtype discriminator.)