CS x265 Exam 3 Spring 2017 Name: ______________KEY________________

ALL questions 5 points, except question 11

I will not use a source other than my brain on this exam: ____________________________________ (please sign)

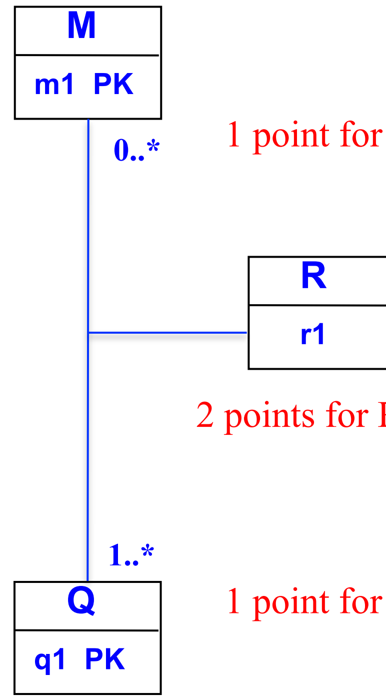
1. (5 points) Consider the following definitions, and give a UML diagram (on the right of the page) that is consistent with the definitions.

```
CREATE TABLE M (
  m1, PRIMARY KEY (m1)
)

CREATE TABLE Q (
   q1, PRIMARY KEY (q1)
)

CREATE TABLE R (
   r1, m1, q1, PRIMARY KEY (m1, q1),
   FOREIGN KEY (m1) REFERENCES M,
   FOREIGN KEY (q1) REFERENCES Q
)

CREATE ASSERTION MparticipatesQ
CHECK (NOT EXISTS
              (SELECT *
               FROM M
               WHERE M.m1 NOT IN
                  (SELECT m1 FROM R)))
```



1 point for 0..*

1 point for both (not each)
M and Q classes

2 points for R

1 point for 1..*

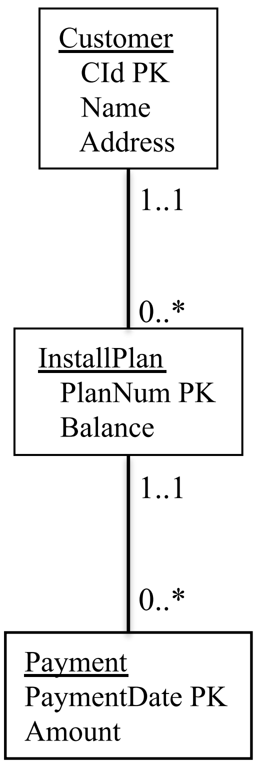
5 minutes

PlanNum is PK and associated 1..1 w Cust

2. (5 points) Consider a DB of a retailer that sells items to customers on an installment plan. The following constraints should hold.

- A customer is identified by a unique identifier (CId) and has an associated Name and Address.
- Each installment plan is identified by a plan number (which is unique across ALL customers), and has the current balance.
- A customer can have zero or more installment plans.
- A **complete history of payments** is recorded, giving the payment date and payment amount for each payment on each plan.
- There are never two payments for **the same plan** recorded for the same date.

a) Briefly state which of the constraints above, if any, that this UML violates (unless you hacked something ugly up to make it work as is):

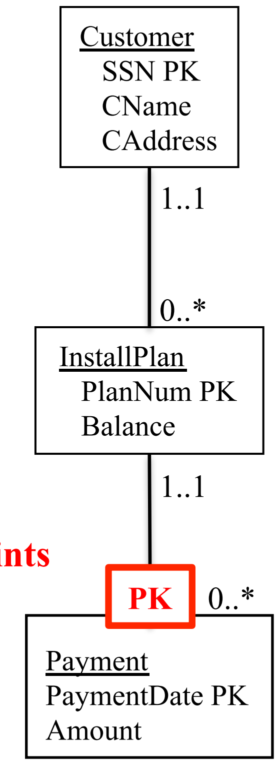


+2 points

The essential problem is that PaymentDate by itself is an insufficient key for Payment, excluding the possibility of *a complete history of payments* across all plans (i.e., no record of payments to different plans on the same date could be made).

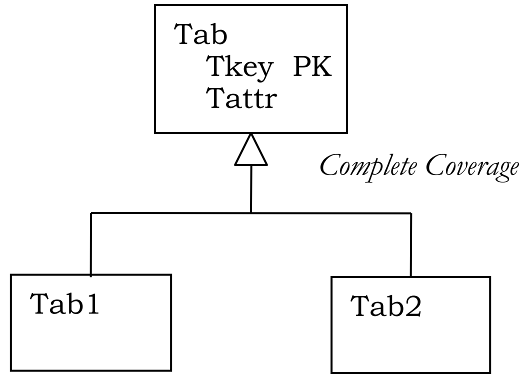
b) Give a UML that satisfies all constraints stated above by making one simple addition to the UML on the left.

+3 points



5 minutes

3. (5 pts) Circle all options that would correctly enforce a *Complete Coverage* constraint of Tab (with subclasses Tab1 and Tab2) in an SQL translation of the following UML fragment.



Intended and initially applied grading scheme:

3 points for one;
4 points for two;
5 points for three;
-2 for (a)

But considering the typo noted below, I am regrading this question. In the regrade, no one's points for question 3 will go down. The parens on choice c are easy to miss because of the formatting, and it wasn't my intent to trick anyone.

a) CREATE ASSERTION CompleteCoverageOfTab1AndTab2
CHECK (SELECT COUNT (DISTINCT Tab.Tkey) FROM Tab)
= (SELECT COUNT (DISTINCT Tab1.Tkey) FROM Tab1)
+ (SELECT COUNT(DISTINCT Tab2.Tkey) FROM Tab2)

b) CREATE ASSERTION CompleteCoverageOfTab1AndTab2
CHECK (NOT EXISTS (SELECT Tab.Tkey FROM Tab
EXCEPT
SELECT Tab1.Tkey FROM Tab1
EXCEPT
SELECT Tab2.Tkey FROM Tab2))

The (…) in red of option (c) were omitted on the exam and since EXCEPT and UNION have equal precedence, and evaluated left to right, option (c) as given in your exam, would not yield the correct answer. At least one student was also misled into thinking that the order of evaluation might be right to left in option (b) because of the typo in (c) – that's unbelievably good eyes (and in some languages, parsing is such that it might be right to left, but that's not typical).

c) CREATE ASSERTION CompleteCoverageOfTab1AndTab2
CHECK (NOT EXISTS (SELECT Tab.Tkey FROM Tab
EXCEPT
(SELECT Tab1.Tkey FROM Tab1 UNION SELECT Tab2.Tkey FROM Tab2)))

d) CREATE ASSERTION CompleteCoverageOfTab1AndTab2
CHECK (NOT EXISTS (SELECT Tab.Tkey FROM Tab
WHERE Tab.Tkey NOT IN (SELECT Tab1.Tkey FROM Tab1) AND
Tab.Tkey NOT IN (SELECT Tab2.Tkey FROM Tab2)

e) None of the above – the primary keys for Tab1 and Tab2 are not specified.
0 points for this

6 minutes

4. (5 points) Consider the following four table definitions, together with all entries in each of the four tables.

```
CREATE TABLE Customer (
  SSN Integer, ...
  PRIMARY KEY (SSN));

CREATE TABLE Product (
  ProdID Integer, ...
  PRIMARY KEY (ProdId));

CREATE TABLE Account (
  SSN Integer NOT NULL,
  AccntNo Integer, ...
  PRIMARY KEY (AccntNo),
  FOREIGN KEY (SSN)
    REFERENCES Customer
    ON UPDATE CASCADE);

CREATE TABLE Transaction (
  TransID Integer,
  AccntNo Integer,
  ProdId Integer, ...
  PRIMARY KEY (TransID),
  FOREIGN KEY (AccntNo)
    REFERENCES Account
    ON UPDATE NO ACTION, /* aka RESTRICT */
  FOREIGN KEY (ProdId)
    REFERENCES Product
    ON UPDATE CASCADE);
```

Customer

SSN	...	
Ssn1	...	
~~Ssn2~~**Ssn5**	...	+1pt
Ssn3	...	

Product

ProdID	...
Pid1	...
Pid2	...
~~Pid3~~ **Pid4**	...

Account

	SSN	AccntNo	...
	Ssn1	Acct1	...
	~~Ssn2~~**Ssn5**	Acct4	...
+1pt	Ssn1	Acct2	...
	~~Ssn2~~ **Ssn1**	Acct3	...
	~~Ssn2~~**Ssn5**	Acct5	...
No change	Ssn3	Acct6	...

+1pt

-2pt if this changed;
-1 for any other change

Transaction

TransID	AcctNo	ProdID	...
Tid1	Acct6	~~Pid3~~**Pid4**	...
Tid2	~~Acct3~~ **Acct4**	Pid2	...
Tid3	Acct3	~~Pid3~~**Pid4**	...

+1pt

+1pt

Change all attribute values as a result of performing these UPDATE operations in order (BE NEAT!!!). If an operation fails, and has no effect as a result, then move to the next operation.

UPDATE Transaction SET AccntNo = Acct4 WHERE TransID = Tid2;
UPDATE Account SET AccntNo = Acct1 WHERE AccntNo = Acct5; violates PK, fails
UPDATE Account SET SSN = Ssn1 WHERE AccntNo = Acct3;
UPDATE Product SET ProdID = Pid4 WHERE ProdID = Pid3; cascades to Trans
UPDATE Customer SET SSN = Ssn5 WHERE SSN = Ssn2; cascades to Account
UPDATE Account SET AccntNo = Acct7 WHERE SSN = Ssn3; blocked in Trans

10 minutes

5. (5 points) Consider the following table definition:

CREATE TABLE RelC (Cid integer, c1 integer, c2 integer, c3 integer, PRIMARY KEY (Cid))

Circle all queries below that are equivalent to the query:

```
SELECT C.c2, AVG (C.c3) AS avc3
FROM RelC C
WHERE C.c3 > 5
GROUP BY C.c2
HAVING COUNT (*) > 1
```

+3 for one right,
+5 for two right

By equivalent, we mean "would return the same result", without concern for efficiency or elogance.

(a)

```
SELECT C.c2, AVG (C.c3) AS avc3
 FROM RelC C
 WHERE C.c3 > 5 AND COUNT(*) > 1
 GROUP BY C.c2
```

- 2pts

(b)

```
SELECT C.c2, AVG (C.c3) AS avc3
FROM RelC C
WHERE C.c3 > 5
GROUP BY C.c2
HAVING 1 < (SELECT COUNT (*)
            FROM RelC C2
            WHERE C.c2 = C2.c2 AND C2.c3 > 5)
```

(c)

```
SELECT C.c2, AVG (C.c3) AS avc3
 FROM RelC C
 GROUP BY C.c2
 HAVING COUNT(*) > 1 AND C.c3 > 5
```

- 2pts

(d)

```
SELECT Temp.c2, Temp.avc3
 FROM (SELECT C.c2, AVG (C.c3) AS avc3, COUNT (*) AS c2cnt
          FROM RelC C
          WHERE C.c3 > 5
          GROUP BY C.c2) AS Temp
 WHERE Temp.c2cnt > 1
```

(e) None of the above

0 total

6 minutes

6. (5 points) Consider the two tables below. Write a CREATE VIEW statement that lists the average water readings for each building of each day, but only for daily averages computed over more than 2 values. The view, call it **Maintenance**, should list ReadingDate, BuildingName, and the average reading for that date/building, listed as AverageValue.

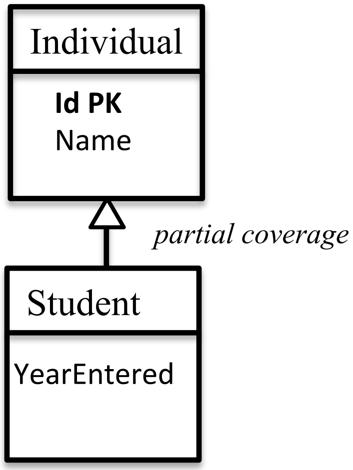
```
CREATE TABLE WaterSensor (
BuildingName VARCHAR(35) NOT NULL,
WaterSensorID INTEGER,
WaterSensorOnLineDate DATE,
PRIMARY KEY (WaterSensorID));
```

```
CREATE TABLE WReading (
WaterSensorID INTEGER,
WReadingDate DATE,
WReadingTime TIME,
WValue INTEGER NOT NULL,
PRIMARY KEY (WaterSensorID, WReadingDate, WReadingTime),
FOREIGN KEY (WaterSensorID) REFERENCES WaterSensor);
```

```
CREATE VIEW Maintenance (ReadingDate, BuildingName, AverageValue ) AS
    SELECT WR.WReadingDate, WS.BuildingName,  AVERAGE(WR.HRWReadingValue) AS AverageValue
    FROM WaterSensor WS, WReading WR
    WHERE WS.WaterSensorID = WR.WaterSensorID
    GROUP BY WR.WReadingDate, WS.BuildingName
    HAVING COUNT(*) > 2
```

5 minutes

7. (5 points) Consider the following UML snippet below. Assume that the two classes are translated into two tables following the usual translation rules for subclasses and parents. Assume further that a VIEW is defined that gives all the attributes of Student (undoubtedly there would be many more than I have included here), to include those that are inherited from Individual. Write an INSTEAD OF TRIGGER that implements INSERTs to WholeStudentView by inserting into the relevant base tables.




```
CREATE VIEW WholeStudentView (Id, Name, YearEntered)
AS SELECT I.Id, I.Name, S.YearEntered
     FROM Individual I, Student S
     WHERE I.Id = S.Id;
```

```
CREATE TRIGGER InsertIntoWholeStudentView
INSTEAD OF INSERT ON WholeStudentView
FOR EACH ROW /* implied by SQLite */
BEGIN
   INSERT INTO Individual VALUES (NEW.Id, NEW.Name);
   INSERT INTO Student VALUES (NEW.Id, NEW.YearEntered);
END;
```

Finish the trigger

+3 for one of these INSERTs; +5 for both

5 minutes

8. (5 points) Consider the relational schema, R(C S J D P Q V K) with functional dependencies (FDs)

J,P → C
S,D → P
J → S
C → S,J,D,P,Q,V

K is not on RHS of any FD;
So K must be part of any key

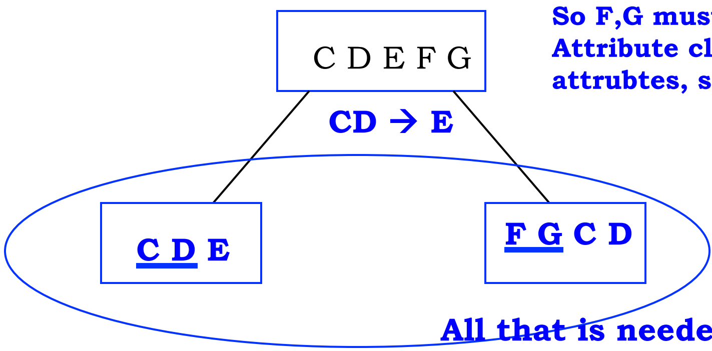
3 pt for one, 4 for two, 5 for all three
-1 for each of any others

Key(s): JPK, CK, JDK

Give *all* minimal keys for R.

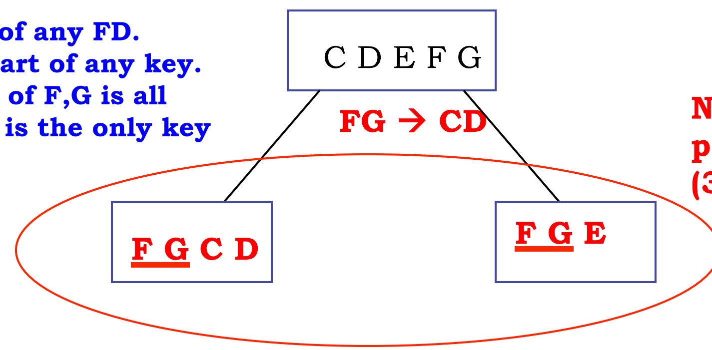
9. (5 points) Consider the relation P with 5 attributes, P(C D E F G) with FDs **C,D → E** and **F,G → C,D**.

Give a dependency-preserving decomposition of P, where each relation of the decomposition is in BCNF. Your decomposition should have as few relations as possible, while still satisfying the specifications of the problem.



F, G not on RHS of any FD.
So F,G must be part of any key.
Attribute closure of F,G is all
attrubtes, so F,G is the only key

All that is needed
in ellipse (5 pts)



Not dependency
preserving
(3 pt only)

Need not show the decomposition tree

10 minutes

10. (5 points) Consider the relational schema R(A, B, C, D, E, F) with functional dependencies

A → F
A,C → B
D → E
A → C
B → F

Give a minimal set of FDs that is informationally equivalent to this set. If the set is already a minimal set, then say so. BE CLEAR!

1. Can LHS of any FD be simplified?
A,C can be simplified because C can be inferred from A, so have both A,C on LHS is redundant. A,C → B can be replaced by A→B

2. Consider FDs in left-to-right order given: {A → F, A → B, D → E, A → C, B → F}.
Can F be inferred from A without A→F? YES {A} → {A,B} → {A,B,C} → {A,B,C,F} So, remove A→F
obtaining {A → B, D → E, A → C, B → F}
Can B be inferred from A without A→B (and without A→F)? No, {A} → {A,C}, so keep A→B
Can E be inferred from D without D→E (and without A→F)? No, {D} → {D}, so keep D→E
Can C be inferred from A without A→C (and without A→F)? No, {A} → {A}, so keep A→C
Can F be inferred from B without B→F (and without B→F)? No, {B} → {B, F} keep B→F

5 points for {A→B, D→E, A→C, B→F } or {A→B,C; D→E; B→F }

-2 if A,C → B is still present
-2 if A→ F is still present

6 minutes

11. (10 points) Consider the relation R(A, B, C, D) with functional dependencies (FDs)

C → A

A → D

D → C

Circle all true statements.

+3 for circling one right,
+4 for two right
+5 for three right,
+6 for four right
+7 for five right,
+8 for six right
+9 for seven right
+10 for seven right

-2 for each incorrect circled

a) R is in BCNF. all three FDs have left hand sides that aren't keys of R

b) R has exactly 3 minimal keys. A,B and B,C and B,D

c) R1(A, D) and R2 (A, B, C) is a lossless decomposition of R. It uses the standard decomposition procedure that ensures lossless (videos, class), decomposing on A→D (i.e., all attributes determinable: D from A in R1 and A,B and B,C are both keys of R2)

d) R1(A, D) and R2 (A, B, C) is a dependency preserving decomposition of R. C→A, A→C, A→D, D→A is an alternative minimal set (of the closure of the stated FDs in the problem), and all of these assignable to R1 or R2

e) Each of R1(A, D) and R2(A, B, C) are in BCNF, where R1 and R2 are a decomposition of R. A,B and B,C are keys, and C→A violates BCNF condition

f) The three FDs given in the statement of this problem are a minimal set (i.e., no proper subset of the three has the same FD closure).

g) Each of R3(A, D), R4(A, C), and R5(B, C) are in BCNF, where R3, R4, and R5 is a decomposition of R.

h) R3(A, D), R4(A, C), and R5(B, C) is a dependency preserving decomposition of R. C→A, A→C, A→D, D→A is an alternative minimal set, and all of these assignable to R3 or R4

i) R6 (A, D), R7(A, C), R8(B, C), and R9(D, C) is a dependency preserving decomposition of R. All FDs in the minimal set given are assignable

j) R10(A, D) and R11(B, C, D) is a lossless decomposition of R. D→A follows from D→C and C→A. If we decompose using the standard procedure that guarantees a lossless decomposition using D→A, we get R10 and R11

6 minutes