Some questions have +I and –I for different options. The intent is that you sum over all options selected by a student, but in all cases, if the sum is less than 0, then make it 0.
1. Consider the relation R with 7 attributes, R = [ A B C D E F ]. You are given the following functional dependencies: Q = {A \rightarrow B, CD \rightarrow E, CF \rightarrow AB}.

(a) (2 pts) There is only one key for R. What is it? **CDF**

CDF does not appear on RHS of any FD; must be part of any key. \{CDF\} \rightarrow \{CDEF\} \rightarrow \{ABCDEF\}

(b) (2 pts) Give a minimal set of FDs equivalent to Q. If Q is already a minimal set, then say so.

Can LHS of any FD be simplified? Can B be inferred from A without A \rightarrow B? No – keep A \rightarrow B
CD can’t be simplified; C cannot be inferred from D, or vice versa Can E be inferred from CD without CD \rightarrow E? No – keep CD \rightarrow E
CF can’t be simplified; C cannot be inferred from F, or vice versa Can A be inferred from CF without CF \rightarrow A? No – keep CF \rightarrow A

Minimal set in green so CF \rightarrow B redundant

2. (2 pts) Consider the relation R with 7 attributes, R = [ A B C D E F ]. You are given the following functional dependencies: Q = {A \rightarrow B, B \rightarrow A, C \rightarrow D, CF \rightarrow E, D \rightarrow E, E \rightarrow F}. Give a minimal set of FDs equivalent to Q. If Q is already minimal, then say so.

Can LHS of any FD be simplified? No other simplifications can be made (without losing information):
CF CAN BE simplified; F can be inferred from C

Same as Q, but WITHOUT CF \rightarrow E

C \rightarrow E is redundant, because

C \rightarrow D \rightarrow E
3. (2 pts) Consider the relation \( P \) with 5 attributes, \( R = \{ C, D, E, F, G \} \). You are given the following functional dependencies: \( Q = \{ CD \rightarrow E, FG \rightarrow CD \} \). 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.

All that is needed in ellipse (2 pts)

FG is the only key

FG \( \rightarrow \) CD

Not dependency preserving (1 pt only)

0 pts otherwise)

4. (5 pts) Consider the relational schema \( [ A, B, C, D, E, F, G ] \) with Functional Dependencies

\[
\begin{align*}
AB & \rightarrow CD, \\
D & \rightarrow EF, \\
CF & \rightarrow G, \\
FG & \rightarrow A
\end{align*}
\]

Give all keys for this relational schema.

- \( B \) must be part of any key, not on rhs of any FD
- \( \{AB\} \rightarrow \{ABCD\} \rightarrow \{ABCDEF\} \rightarrow \{ABCDEFG\} \)
- \( \{ABG\} \rightarrow \{ABFG\} \rightarrow \{ABCDFG\} \rightarrow \{ABCDEFG\} \)
- \( \{BCF\} \rightarrow \{BCFG\} \rightarrow \{ABCFG\} \rightarrow \{ABCDFG\} \rightarrow \{ABCDEFG\} \)
- \( \{BCD\} \rightarrow \{BCDEFG\} \rightarrow \{ABCDEFG\} \)

\( AB, BFG, BCF, BCD, BDG \)
5. (2 pts) Consider the relational schema \([ A \ B \ C \ D ]\) with FDs \(F = \{ A \rightarrow B, B \rightarrow CD, CD \rightarrow A \}\); \(F\) is a minimal set

Give an ALTERNATE minimal set, \(G\), that is equivalent to \(F\) (e.g., the same FDs follows from \(G\) as from \(F\)). Note, that \(F\) and \(G\) can share some FDs in common, but they should not be identical sets.

6. (1 pt) Give an SQL command that would grant a user named SMITH, SELECT and UPDATE privileges on table Products with a GRANT option.

   \[\text{GRANT SELECT, UPDATE ON Products TO SMITH with GRANT option (or something close)}\]

7. (1 pt) Name the ACID property that is facilitated by the SQL \texttt{ROLLBACK} command? \textbf{Atomicity}
8. A colleague brings you three table definitions, summarized by these relational schema (R, P, Q), with ‘a’ as a primary key for table R, ‘b’ the primary key for table P, and ‘c’ the primary key for table Q. ‘b’ is a foreign key from R to P, and ‘c’ is a foreign key from P to Q.

\[
\begin{array}{ccc}
R & \rightarrow & P \\
\downarrow & & \downarrow \\
\text{a} & \text{b} & \text{b} & \text{c} & \text{c} & \text{d} \\
\text{R} & \text{P} & \text{Q}
\end{array}
\]

In addition to the table definitions, your colleague gives you this assertion, intended to enforce the FD Q.d \(\rightarrow\) R.a.

```
CREATE ASSERTION AsPerD
CHECK (NOT EXISTS (SELECT *
FROM (SELECT COUNT (DISTINCT R.a) AS cnt
FROM R, P, Q
WHERE R.b = P.b AND P.c = Q.c
GROUP BY Q.d, R.a) AS Temp
WHERE Temp.cnt > 1))
```

**(a) (2 pts)** Ignoring for the moment that your colleague requires a course in DB design, you recognize that the assertion is incorrect, but that you can correct it by making ONE simple STRIKETHROUGH. Put a line through that part of the assertion definition so that the corrected version properly enforces the FD, d \(\rightarrow\) a.

**(b) (2 pts)** After you explain your fix, your colleague leaves, and you replace your colleague’s three tables and one (corrected) assertion by ONE table definition that enforces all the constraints encoded in the original three tables and one assertion. Give the definition for this one table as a CREATE TABLE statement.

```
CREATE TABLE ABCD (
    a, b, c, d
PK (a)
UNIQUE(b), UNIQUE(c), UNIQUE(d)
)
```

Any of a,b,c,d could be the PK, others UNIQUE.
9. (4 points) Consider the following two transactions, T1 and T2:

T1: Read(A), $\text{Op}_{11}(A)$, Write(A), Read(B), $\text{Op}_{12}(B)$, Write(B), Commit

T2: Read(A), $\text{Op}_{21}(A)$, Write(A), Read(B), $\text{Op}_{22}(B)$, Write(B), Commit

Circle all schedules (just showing disk reads and writes) that clearly result in *serializable* behavior even without knowing when the Ops are performed.

+2 for one, + 3 for two, +4 for three
-1 for one incorrect circled, -2 for two,
-4 for three
0 min, 4 max

(G) NONE OF THE ABOVE

0 total for question if this circled
**10. (3 pts)** Circle all options that would correctly enforce the No Overlap (aka Disjoint) constraint between Tab1 and Tab2 in an SQL translation of the following UML fragment.

-1 for each option (a) or (c) (again, a negative score will simply be counted Zero/0);

1 point for each

a) CREATE ASSERTION NoOverlapBetweenTab1AndTab2
CHECK ((SELECT COUNT (Tab1.Tkey) FROM Tab1) = (SELECT COUNT (Tab2.Tkey) FROM Tab2))

b) CREATE ASSERTION NoOverlapBetweenTab1AndTab2
CHECK (NOT EXISTS (SELECT Tab1.Tkey FROM Tab1) INTERSECT (SELECT Tab2.Tkey FROM Tab2))

c) CREATE ASSERTION NoOverlapBetweenTab1AndTab2
CHECK (EXISTS (SELECT * FROM Tab1 T1, Tab2 T2 WHERE T1.Tkey = T2.Tkey))

d) CREATE ASSERTION NoOverlapBetweenTab1AndTab2
CHECK (NOT EXISTS (SELECT * FROM Tab1 T1, Tab2 T2 WHERE T1.Tkey = T2.Tkey))

e) CREATE ASSERTION NoOverlapBetweenTab1AndTab2
CHECK (NOT EXISTS (SELECT Tab1.Tkey FROM Tab1
WHERE Tab1.Tkey IN (SELECT Tab2.Tkey FROM Tab2)
UNION
SELECT Tab2.Tkey FROM Tab2
WHERE Tab2.Tkey IN (SELECT Tab1.Tkey FROM Tab1)))

f) None of the above 0 points if this is circled, with or without other choices
11. (4 pts) Consider the UML fragment to the right and identify (circle) **all** equivalent table translations (i.e., those translations that faithfully enforce the constraints implied by the UML without regard to elegance) from those given below. You might receive partial credit for a brief explanation of your choices. `UNIQUE(y)` implies that `y` NOT NULL, but not vice versa. PK stands for PRIMARY KEY. FK stands for FOREIGN KEY.

<table>
<thead>
<tr>
<th>(A)</th>
<th>(B) 2 points</th>
<th>(C) 2 points</th>
<th>(D)</th>
<th>(E) -2 points</th>
<th>(F) None of the above</th>
</tr>
</thead>
<tbody>
<tr>
<td>CREATE TABLE X (x1, PK (x1), FK (x1) refs R)</td>
<td>CREATE TABLE XR (x1, r1, z1, PK(x1), FK (z1) refs Z)</td>
<td>CREATE TABLE XR (x1, r1, z1 NOT NULL, PK(x1), FK (z1) refs Z)</td>
<td>CREATE TABLE XR (x1, r1, z1 NOT NULL, PK(x1), FK (z1) refs Z)</td>
<td>CREATE TABLE Z (z1, PK(z1))</td>
<td></td>
</tr>
<tr>
<td>CREATE TABLE R (x1, r1, z1 NOT NULL, PK(x1), FK (z1) refs Z, FK (x1) refs X)</td>
<td>CREATE TABLE Z (z1, PK(z1))</td>
<td>CREATE TABLE Z (z1, PK(z1))</td>
<td>CREATE TABLE Z (z1, PK(z1))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CREATE TABLE Z (z1, PK (z1))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

-1 points 2 points -2 points -2 points

0 points if circled, with or without other choices.
12. (4 pts) Consider the UML fragment to the right and identify (circle) all equivalent table translations (i.e., those translations that faithfully enforce the constraints implied by the UML without regard to elegance) from those given below. You might receive partial credit for a brief explanation of your choices. UNIQUE(y) implies that y NOT NULL, but not vice versa. PK stands for PRIMARY KEY. FK stands for FOREIGN KEY.

(A)  
CREATE TABLE X ( 
  x1, 
  PK (x1) 
) 
CREATE TABLE R ( 
  x1, r1, z1, 
  PK(x1, z1), 
  FK (z1) refs Z, 
  FK (x1) refs X 
) 
CREATE TABLE Z ( 
  z1, 
  PK (z1) 
) 
CREATE ASSERTION 
XparticipatesZ 
CHECK 
(NOT EXISTS 
(SELECT * FROM X 
WHERE  X.x1 NOT IN 
(SELECT Z.x1 FROM Z)))

(B)  
CREATE TABLE X ( 
  x1, 
  PK (x1) 
) 
CREATE TABLE Z ( 
  x1, r1, z1, 
  PK(z1), 
  FK (x1) refs X 
) 
CREATE ASSERTION 
XparticipatesZ 
CHECK 
(NOT EXISTS 
(SELECT * FROM X 
WHERE  X.x1 NOT IN 
(SELECT Z.x1 FROM Z)))

(C) 2 points  
CREATE TABLE X ( 
  x1, 
  PK (x1) 
) 
CREATE TABLE R ( 
  x1 NOT NULL, r1, z1, 
  PK(z1), 
  FK (z1) refs Z, 
  FK (x1) refs X 
) 
CREATE TABLE Z ( 
  z1, 
  PK (z1) 
) 
CREATE ASSERTION 
XparticipatesZ 
CHECK 
(NOT EXISTS 
(SELECT * FROM X 
WHERE  X.x1 NOT IN 
(SELECT Z.x1 FROM Z)))

(D) 2 points  
CREATE TABLE X ( 
  x1, 
  PK (x1) 
) 
CREATE TABLE R ( 
  x1 NOT NULL, r1, z1, 
  PK(z1), 
  FK (z1) refs Z, 
  FK (x1) refs X 
) 
CREATE TABLE Z ( 
  z1, 
  PK (z1) 
) 
CREATE ASSERTION 
XparticipatesZ 
CHECK 
(NOT EXISTS 
(SELECT * FROM X 
WHERE  X.x1 NOT IN 
(SELECT Z.x1 FROM Z)))

(E) 0 points  
CREATE TABLE X ( 
  x1, 
  PK (x1) 
) 
CREATE TABLE Z ( 
  x1, r1, z1, 
  PK(z1), 
  FK (x1) refs X 
) 
CREATE ASSERTION 
XparticipatesZ 
CHECK 
(NOT EXISTS 
(SELECT * FROM X 
WHERE  X.x1 NOT IN 
(SELECT Z.x1 FROM Z)))

(F) None of the others  
0 points if circled, with or without other choices
13. **(4 pts)** Consider the following table definitions:

CREATE TABLE RelA (Akey integer, a1 integer, a2 integer, a3 integer, PRIMARY KEY (Akey))
CREATE TABLE RelB (Bkey1 integer, Bkey2 integer, b1 integer, PRIMARY KEY (Bkey1, Bkey2), FOREIGN KEY (Bkey1) REFERENCES RelA (Akey))

Circle all queries below that are equivalent to the query: SELECT A.a2, A.a3
FROM RelA
WHERE A.Akey IN (SELECT B.Bkey1
FROM RelB
WHERE A.a2 = B.Bkey2 AND A.a1 = B.b1)

(a) SELECT A.a2, A.a3 FROM RelA
WHERE EXISTS (SELECT *
FROM RelB
WHERE A.Akey = B.Bkey1
AND A.a1 = B.b1
AND A.a2 = B.Bkey2)

(b) SELECT A.a2, A.a3 FROM RelA, RelB
WHERE A.Akey = B.Bkey1 AND A.a1 = B.b1 AND A.a2 = B.Bkey2

(c) SELECT C.c2, AVG (C.c3) AS ave3
FROM RelC C
GROUP BY C.c2
HAVING COUNT(*) > 1 AND C.c3 > 5

(d) SELECT Temp.c2, Temp.ave3
FROM (SELECT C.c2, AVG (C.c3) AS ave3, COUNT(*) AS c2count
FROM RelC C
WHERE C.c3 > 5
GROUP BY C.c2) AS Temp
WHERE Temp.c2count > 1

(e) SELECT Temp.t1, Temp.t2
FROM (SELECT Bkey1 AS t1, a3 AS t2
FROM RelA, RelB
WHERE Akey = Bkey1 AND a1 = b1
AND a2 = Bkey2)
AS Temp

(f) None of the others

If this had been Bkey2 then this would be correct; I didn’t take off if someone did mark it, as it was at a level of detail that I didn’t intend to test
14. (3 pts) Consider the following table definitions:

CREATE TABLE RelA (Aid integer, a1 integer, a2 integer, PRIMARY KEY (Aid))
CREATE TABLE RelB (Aid integer, Cid integer, b1 integer, PRIMARY KEY (Aid, Cid, b1), FOREIGN KEY (Aid) REFERENCES RelA, FOREIGN KEY (Cid) REFERENCES RelC)
CREATE TABLE RelC (Cid integer, c1 integer, c2 integer, c3 integer, PRIMARY KEY (Cid))

Circle all queries below that are equivalent to:

\[ \pi_{c1} (( Temp1 \cap Temp2 ) \bigtriangleup RelC) \]
where

Temp1 = \pi_{Cid} (( \sigma_{a2=q} RelA ) \bigtriangleup RelB) and
Temp2 = \pi_{Cid} (( \sigma_{a2=r} RelA ) \bigtriangleup RelB)

By equivalent, we mean produces the same output given the same input (and we are not referring to efficiency or elegance).

Total cannot exceed 5 and cannot be less than 0

(a) SELECT DISTINCT C.c1
FROM RelC C, RelB B1, RelA A1, RelB B2, RelA A2

+1 pt

(b) ) SELECT DISTINCT C.c1
FROM RelC C, RelB B, RelA A
WHERE C.Cid = B.Cid AND B.Aid = A.Aid AND A.a2 = q
INTERSECT
SELECT DISTINCT C2.c1
FROM RelC C2, RelB B2, RelA A2

-1 pt

(c) SELECT DISTINCT C.c1
FROM RelA A, RelB B, RelC C
WHERE C.Cid = B.Cid AND B.Aid = A.Aid AND A.a2 = q AND C.Cid IN (SELECT C2.Cid
FROM RelC C2, RelA A2, RelB B2

+1 pts

(d) SELECT DISTINCT C.c1
FROM RelC C
WHERE C.Cid IN (( SELECT B.Cid
FROM RelA A, RelB B
WHERE B.Aid = A.Aid AND A.a2 = q)
INTERSECT
(( SELECT B2.Cid
FROM RelA A2, RelB B2
WHERE B2.Aid = A2.Aid AND A2.a2 = r))

+1 pts

(e) \pi_{c1} (Temp1 \bigtriangleup RelB \bigtriangleup RelC) where Temp1 = (\sigma_{a2=q} RelA) \cap (\sigma_{a2=r} RelA)

-2 pt

(f) None of the above

0 pts total
15. **(3 points)** Consider the two relational schema, SV and VMo, with specific instances, \( SV_i \) and \( VMo_i \):

FOR THE EXAMPLE ABOVE, circle all variations on the outer join that would produce \( SVMo_i \).

+1 for each,

-1.5 for each

(f) None of the above 0 total
16. (2 points) Consider the following relational schemas.

Customers (SSN: integer, name: string, address: string, city: string)

Accounts (SSN: integer, AccntNo: integer, balance: real)

Transactions (AccntNo: integer, ProductId: integer, date: string)

Products (ProductId: integer, ProductName: string, cost: integer)

CREATE TABLE Customers (SSN Integer,
Name CHAR[25] NOT NULL,
Address CHAR[25] NOT NULL,
City CHAR[25] NOT NULL,
PRIMARY KEY (SSN))

CREATE TABLE Products (ProductId Integer,
ProductName CHAR[15],
Cost Integer,
PRIMARY KEY (ProductId))

CREATE TABLE Accounts (SSN Integer NOT NULL,
AccntNo Integer,
Balance Float NOT NULL,
PRIMARY KEY (AccntNo),
FOREIGN KEY (SSN) REFERENCES Customers ON DELETE CASCADE)

CREATE TABLE Transactions (AccntNo Integer,
ProdId Integer,
Date CHAR[6],
PRIMARY KEY (AccntNo, ProdId),
FOREIGN KEY (AccntNo) REFERENCES Accounts ON DELETE NO ACTION,
FOREIGN KEY (ProductId) REFERENCES Products ON DELETE NO ACTION)

Consider the CREATE TABLE statements that implement the relations to the left. Complete these definitions so that if a Customer is deleted, all the Customer’s Accounts are deleted (i.e., all Accounts with an SSN that matches the deleted Customer’s SSN), unless one or more of the Customer’s Accounts participates in any Transaction (as identified by AccntNo), in which case the attempt to delete the Customer is blocked. Also, the definitions should insure that a product can only be deleted if it does not participate in any transaction (as identified by ProdId in Transactions). While certain default settings might have otherwise applied in answering this problem, I want you to ignore defaults and make explicit additions to the appropriate definitions.

-1 FOR EACH if not completely right (but forgive minor syntax errors) or missing
17. Using the Products and Transactions definitions of the previous page, consider the following statement

```sql
INSERT INTO Products (ProductID)
SELECT DISTINCT ProdId
FROM Transactions
WHERE ProdId NOT IN (SELECT ProductID FROM Products)
```

You can disregard any angst you might have concerning Foreign Key constraints having to be checked, and when that would happen.

(a) (3 pts) List the authorizations that the executer of this insertion statement must have in order to successfully execute the statement. Separately number each authorization, and give the minimally-scoped authorizations (a minimally scoped authorization is one that allows the process to succeed, but grants no additional rights that are not required by the process).

```
INSERT(Products.ProductID),
SELECT(Transactions.ProdId),
SELECT(Products.ProductID)
```

(b) (1 point) Explain why the following statement would NOT be legal, even with proper authorizations, given the definitions of the previous page

```sql
INSERT INTO Transactions (ProdId)
SELECT DISTINCT ProductID
FROM Products
WHERE ProductID NOT IN (SELECT ProdId FROM Transactions)
```

The PK for Transactions includes AccntNo, which can therefore not be NULL, but this INSERT statement would attempt to set AccntNo to NULL
18. (3 pts) Using the definitions of two pages ago, and repeated here

Accounts (SSN: integer, AccntNo: integer, balance: real)

Transactions (AccntNo: integer, ProductId: integer, date: string, bill: real)

Write a row-level trigger that increments the appropriate (defined by matching AccntNo) balance in the Accounts table by the bill value of a newly inserted transaction to the Transactions table.

CREATE TRIGGER UpdateAccountBalance
AFTER INSERT ON Transactions
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
UPDATE Accounts A
SET balance = (SELECT SUM(bill)
FROM Transactions T
WHERE T.AccntNo = A.AccntNo)
WHERE A.AccntNo = NewRow.AccntNo;

CREATE TRIGGER
AFTER INSERT ON Transactions
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
UPDATE Accounts
SET balance = balance + NewRow.bill
WHERE Accounts.AccntNo = NewRow.AccntNo;

Accepting of close (incorrect) syntactic variants
19. **(2 pts)** Consider the following UML diagram. Make minimal changes to the translation below, so that the translation is consistent with the constraints indicated by the UML diagram. You will not change the number of tables of the translation. Assume that all attributes are integers and do not indicate the attribute types, for reasons of convenience, on the translation. Note that all relationships are qualified by participation and keys constraints.

```
CREATE TABLE W (  
W1,  
W2,  
PRIMARY KEY (W1)  
)  
```

Make minimal changes to this two-table translation so that it is consistent with the UML diagram.

```
CREATE TABLE XRYPZ (  
W1 NOT NULL,  
Z1,  
Z2,  
P1,  
Y1 NOT NULL,  
Y2,  
R1,  
X1,  
X2,  
PRIMARY KEY (X1) ,  
FOREIGN KEY (W1) REFERENCES W  
)
```

Add UNIQUE(W1), UNIQUE(Y1) to table XRYPZ (1 pts for one, 2 points for two)

You might have seen the relevance of this to question 8b, and vice versa.
20. (3 pts) Consider the B+ tree below.

Note that this tree does not show data nodes, and you do not need to see the data nodes to answer this question. At each leaf, N* is an index of the form <N, <page id, slot #>>, where N is the value of the search key.

Show the tree that results from inserting a record with search key 84 (do not use redistribution). If you can do so clearly and unambiguously, then you can circle and label subtrees in this diagram that do not change and use those labels in your answer on the next page.

answer on next page
(will be passed up, and split the root by middle value of 64)
Write resulting B+ tree for (b) here

Only correct tree

-2 if 84 or 81 does not appear at leaf

-2 for any tree that isn’t 4 levels. Use discretion on partial credit.
21. (4 pts) Consider the B+ tree index for attribute A of table T. Above each node is a numeric label for the node (1 through 11), which you will use in answering this question.

For each of the following operations, list the nodes (by label), in proper order, that would be locked (shared or exclusive) in strict two-phase locking (2PL) when performing the respective operation. If no nodes need be locked, then write None. Ignore data nodes, which are not shown, and do not list new nodes that might be introduced. Assume that this index for attribute A is used in evaluating each operation below. Write S(label) for a shared lock, and X(label) for an exclusive lock. Note that the same node, A, may be listed twice, first as S(A) then as X(A), for the same operation. Do not show order of lock release (we’ll assume all locks released at end of operation, upon commit). Treat each operation as independent, and not as a sequence of actions. Assume that redistribution is never used.

(a) SELECT T.A FROM T WHERE T.A > 75 : S(1), S(4), S(10), S(11)

(b) UPDATE T SET T.B = T.B + 100 WHERE T.A = 37 : S(1), S(2), S(6)

(c) UPDATE T SET T.A = T.A + 5 WHERE T.A = 29 : S(1), S(2), S(6), X(6)

(d) INSERT INTO T (A, B, C) VALUES (70, 20, 10) : S(1), S(3), S(9), X(9), X(3), X(1)

subtle: X(8) and/or X(10) too so as to update ptr to 9

In most cases, probably full credit for selected variations, notably skipping over S(k) straight to X(k)
22. (3 pts) Consider the extendible hash table to the left. Assume Hash(x) = x. Show the result of inserting the following keys in order: 41, 22.

-1 for each misplaced key (order within a bin not important)

Answer here
Insert 41

Insert 22

Intermediate answer