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.

**Final Exam (CS 265)** **67 points** **Name:** KEY

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

(For all that follows: a *key* is minimal; no proper subset of a key also functional determines all attributes of a relation)

**1.** Consider the relation R with 7 attributes, R = [ A B C D E F ]. You are given the following functional dependencies:
Q = {A → B, CD → E, CF → 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} → {CDEF} → {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?**

**CD can't be simplified; C cannot be inferred from D, or vice versa**

**CF can't be simplified; C cannot be inferred from F, or vice versa**

**Can B be inferred from A without A→B? No – keep A→B**

**Can E be inferred from CD without CD→E? No – keep CD→E**

**Can A be inferred from CF without CF→A? No – keep CF→A**

**Can B be inferred from CF without CF→B? Yes – CF→A→B, so CF→B redundant**

**Minimal set in green**

**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 → B, B → A, C → D, ~~CF → E~~, D → E, E → 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?**

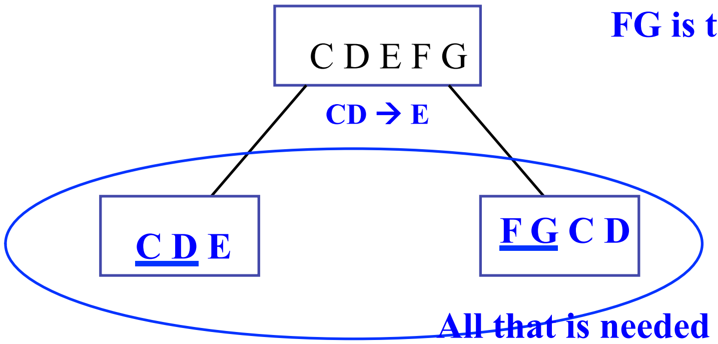
**CF CAN BE simplified; F can be inferred from C**

**C→E is redundant, because C→D→E**

**No other simplifications can be made (without losing information):**

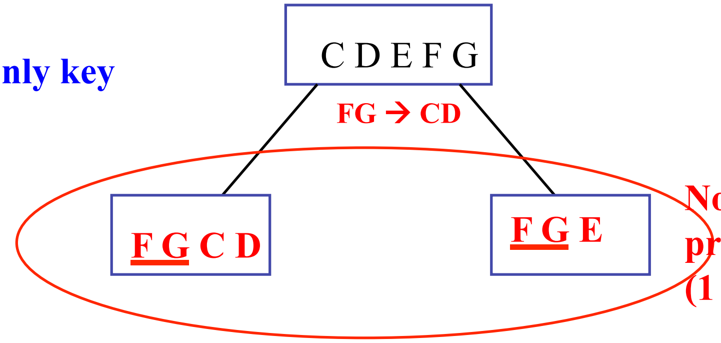
**Same as Q, but WITHOUT CF→E**

**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 → E, FG → 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**



**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

AB→CD,
D→EF,
CF→G,
FG→A

1 pt for each, -1 for any others

Give *all* keys for this relational schema.

**B must be part of any key, not on rhs of any FD**

**{BDG} → {BDEFG} → {ABDEFG} → {ABCDEFG}**

**{AB} → {ABCD} → {ABCDEF} → {ABCDEFG}**
**{BFG} → {ABFG} → {ABCDFG} → {ABCDEFG}**
**{BCF} → {BCFG} → {ABCFG} → {ABCDFG} → {ABCDEFG}**
**{BCD} → {BCDEFG} → {ABCDEFG}**

**AB, BFG, BCF, BCD, BDG**

5. **(2 pts)** Consider the relational schema [ A B C D ] with FDs F = { A→B, B→CD, CD → 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.

**A→B**
**B→C**
**B→D**
~~**CD→A**~~
~~**A→C**~~
~~**A→D**~~
**B→A**
**CD→B**

~~**A→B**~~
~~**B→C**~~
~~**B→D**~~
~~**CD→A**~~
**A→C**
**A→D**
**B→A**
**CD→B**

**A→B**
~~**B→C**~~
~~**B→D**~~
**CD→A**
**A→C**
**A→D**
**B→A**
~~**CD→B**~~

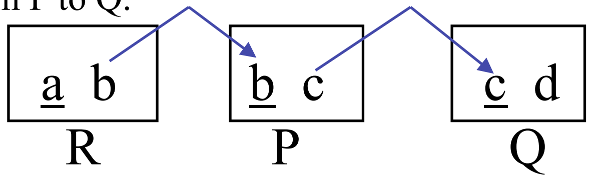
~~**A→B**~~
**B→C**
**B→D**
**CD→A**
**A→C**
**A→D**
~~**B→A**~~
**CD→B**

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.

**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 `ROLLBACK` command? ______**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.



In addition to the table definitions, your colleague gives you this assertion, intended to enforce the FD Q.d → 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 → 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)
)
```

**a→b, b→c, c→d, d→a**

**Thus, b→a, c→b, d→c, a→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), Op$_{11}$(A), Write(A), Read(B), Op$_{12}$(B), Write(B), Commit

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

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

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

[✓ (A)]

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |
| R(B) | |
| W(B) | |
| | R(B) |
| | W(B) |
| Commit | |
| | Commit |

[✓ (B)]

| T1 | T2 |
|---|---|
| | R(A) |
| | W(A) |
| R(A) | |
| | R(B) |
| | W(B) |
| W(A) | |
| R(B) | |
| W(B) | |
| | Commit |
| Commit | |

(C)

| T1 | T2 |
|---|---|
| R(A) | |
| W(A) | |
| | R(A) |
| | W(A) |
| | R(B) |
| | W(B) |
| | Commit |
| R(B) | |
| W(B) | |
| Commit | |

(D)

| T1 | T2 |
|---|---|
| | R(A) |
| | W(A) |
| R(A) | |
| W(A) | |
| R(B) | |
| W(B) | |
| Commit | |
| | R(B) |
| | W(B) |
| | Commit |

[✓ (E)]

| T1 | T2 |
|---|---|
| | R(A) |
| | W(A) |
| | R(B) |
| R(A) | |
| W(A) | |
| | W(B) |
| R(B) | |
| W(B) | |
| | Commit |
| Commit | |

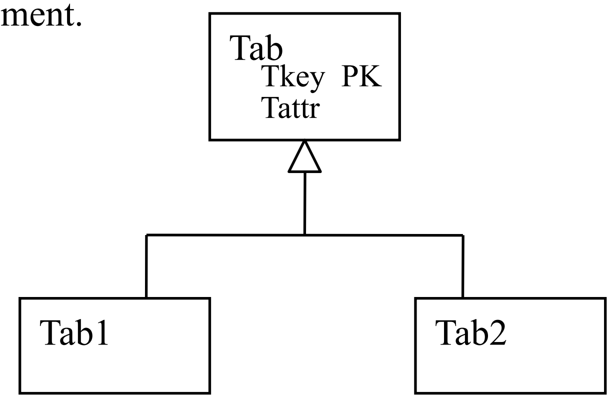
(F)

| T1 | T2 |
|---|---|
| R(A) | |
| | R(A) |
| W(A) | |
| | W(A) |
| | R(B) |
| R(B) | |
| | W(B) |
| W(B)) | |
| | Commit |
| Commit | |

(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) )
```

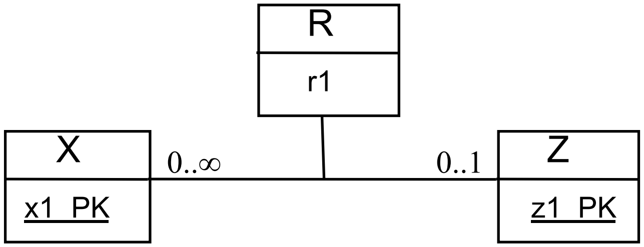
c)
```
CREATE ASSERTION NoOverlapBetweenTab1AndTab2
CHECK (EXISTS (SELECT * FROM Tab1 T1, Tab2 T2 WHERE T1.Tkey  = T2.Tkey))
```

[✓ 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.



(A)

```
CREATE TABLE X (
  x1,
  PK (x1),
  FK (x1) refs R
)

CREATE TABLE R (
   x1, r1,
   z1 NOT NULL,
   PK(x1),
   FK (z1) refs Z,
   FK (x1) refs X
)

CREATE TABLE Z (
   z1,
   PK (z1)
)
```

-1 points

[✓ (B)]

2 points

```
CREATE TABLE X (
  x1,
  PK (x1)
)

CREATE TABLE R (
   x1, r1,
   z1 NOT NULL,
   PK(x1),
   FK (z1) refs Z,
   FK (x1) refs X
)

CREATE TABLE Z (
   z1
   PK (z1)
)
```

[✓ (C)]

2 points

```
CREATE TABLE XR (
   x1, r1, z1,
   PK(x1),
   FK (z1) refs Z
)

CREATE TABLE Z (
   z1,
   PK(z1)
)
```

(D)

```
CREATE TABLE XR (
    x1, r1,
    z1 NOT NULL,
    PK(x1),
    FK (z1) refs Z
)

CREATE TABLE Z (
   z1,
   PK(z1)
)
```

-1 points

(E)

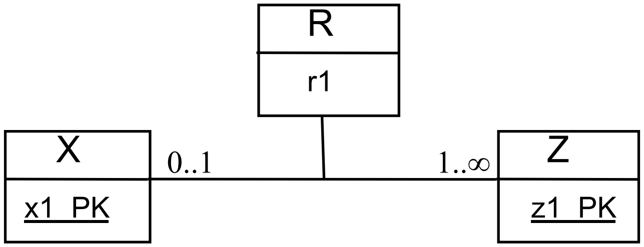
```
CREATE TABLE XR (
   x1, r1, z1,
   PK(x1),
   UNIQUE(z1),
   FK (z1) refs Z
)

CREATE TABLE Z (
   z1,
   PK(z1)
)
```

-2 points

(F) None of the above

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 R.x1 FROM R)))
```

-2 points

(B)

```
CREATE TABLE X (
  x1,
  PK (x1)
)

CREATE TABLE Z (
   x1, r1, z1
   PK(z1),
   FK (x1) refs X
)
```

-2 points

[✓ (C)] 2 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)))
```

2 points [✓ (D)]

```
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 R.x1 FROM R)))
```

(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 A
WHERE A.Akey IN (SELECT B.Bkey1
                 FROM RelB B
                 WHERE A.a2 = B.Bkey2 AND A.a1 = B.b1)
```

[✓ (a)] 1 pt

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

[✓ (b)] 3 pts

```
SELECT A.a2, A.a3
FROM RelA A, RelB B
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~~

- 2pts

(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
```

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

(f) None of the others

0 total

**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: $\pi_{c1}((\text{Temp1} \cap \text{Temp2}) \bowtie \text{RelC})$

where $\text{Temp1} = \pi_{Cid}((\sigma_{a2=q}\,\text{RelA}) \bowtie \text{RelB})$ and

$\text{Temp2} = \pi_{Cid}((\sigma_{a2=r}\,\text{RelA}) \bowtie \text{RelB})$

*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
WHERE C.Cid = B1.Cid AND B1.Aid = A1.Aid AND
          C.Cid = B2.Cid AND B2.Aid = A2.Aid AND
          A1.a2 = q AND A2.a2 = r
```

+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
  WHERE C2.Cid = B2.Cid AND B2.Aid = A2.Aid AND A2.a2 = r
```

-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
                         WHERE C2.Cid = B2.Cid AND
                                   B2.Aid = A2.Aid  AND A2.a2 = r )
```

+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) $\pi_{c1}(\text{Temp1} \bowtie \text{RelB} \bowtie \text{RelC})$

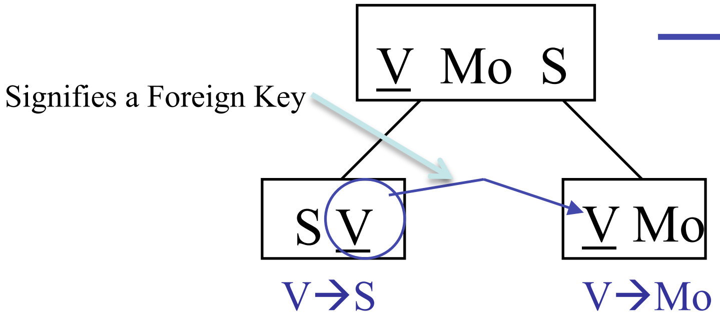
where $\text{Temp1} = (\sigma_{a2=q}\,\text{RelA}) \cap (\sigma_{a2=r}\,\text{RelA})$

-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 $VMo_i$:



$SV_i$

| | |
|---|---|
| S1 | V4 |
| S1 | V5 |
| S2 | V6 |

$VMo_i$

| | |
|---|---|
| V1 | Mo1 |
| V2 | Mo2 |
| V3 | Mo1 |
| V4 | Mo3 |
| V5 | Mo4 |
| V6 | Mo2 |
| V7 | Mo3 |

$SVMo_i$

| S | V | Mo |
|---|---|---|
| NULL | V1 | Mo1 |
| NULL | V2 | Mo2 |
| NULL | V3 | Mo1 |
| S1 | V4 | Mo3 |
| S1 | V5 | Mo4 |
| S2 | V6 | Mo2 |
| NULL | V7 | Mo3 |

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

+1 for each,

[✓ (a)] SV NATURAL RIGHT OUTER JOIN Vmo

[✓ (b)] VMo NATURAL LEFT OUTER JOIN SV

(c) VMo NATURAL RIGHT OUTER JOIN SV

(d) SV NATURAL LEFT OUTER JOIN Vmo

-1.5 for each

[✓ (e)] SV NATURAL FULL OUTER JOIN Vmo

(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)

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.

```
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 Accounts (
  SSN Integer NOT NULL,
  AccntNo Integer,
  Balance Float NOT NULL,
  PRIMARY KEY (AccntNo),

  FOREIGN KEY (SSN)
     REFERENCES Customers
     ON DELETE CASCADE
)
```

**1pt**

```
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)
```

**0.5 pt**

**0.5 pt**

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

**-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

```
 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

```
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.

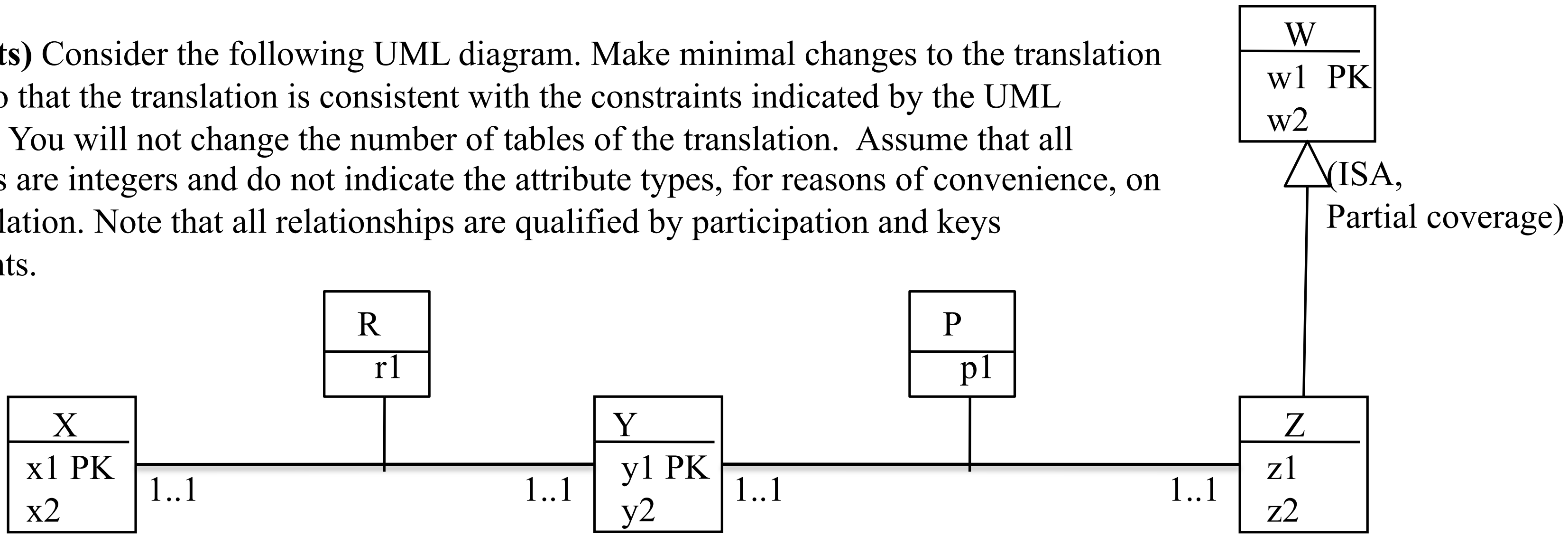
```
CREATE  TRIGGER UpdateAccountBalance
AFTER INSERT ON Transactions    // Complete the Trigger definition
```

```
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;
```

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



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

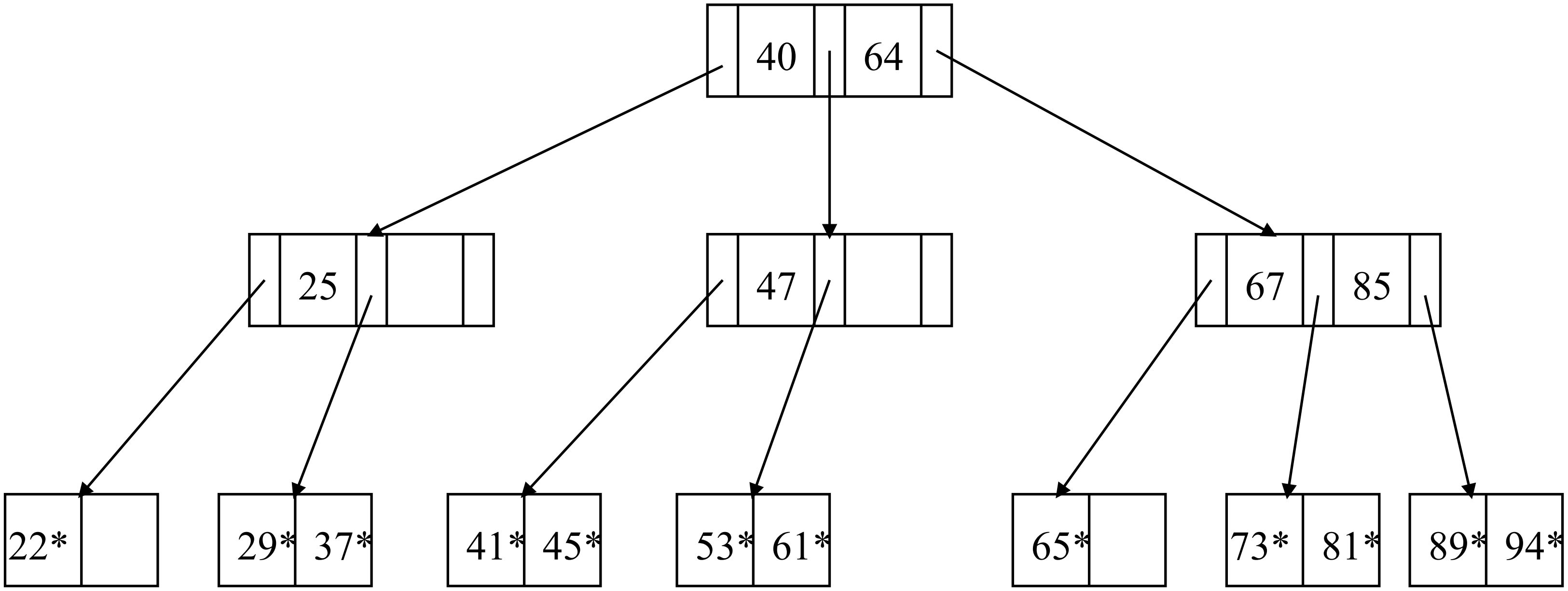
```
CREATE TABLE W (
  W1,
  W2,
  PRIMARY KEY (W1)
)
```

```
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.**

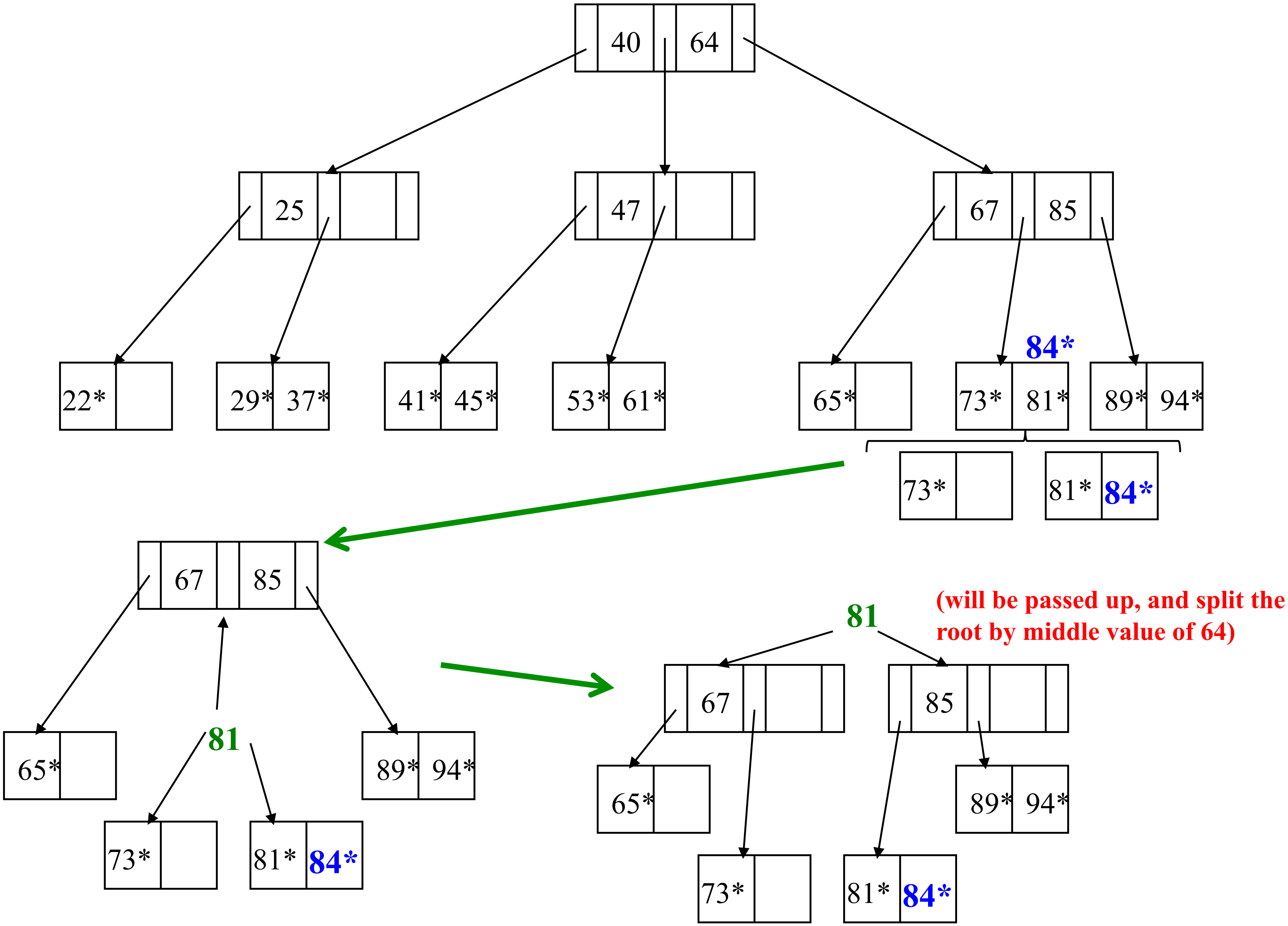
**20. (3 pts)** Consider the B+ tree below.



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.

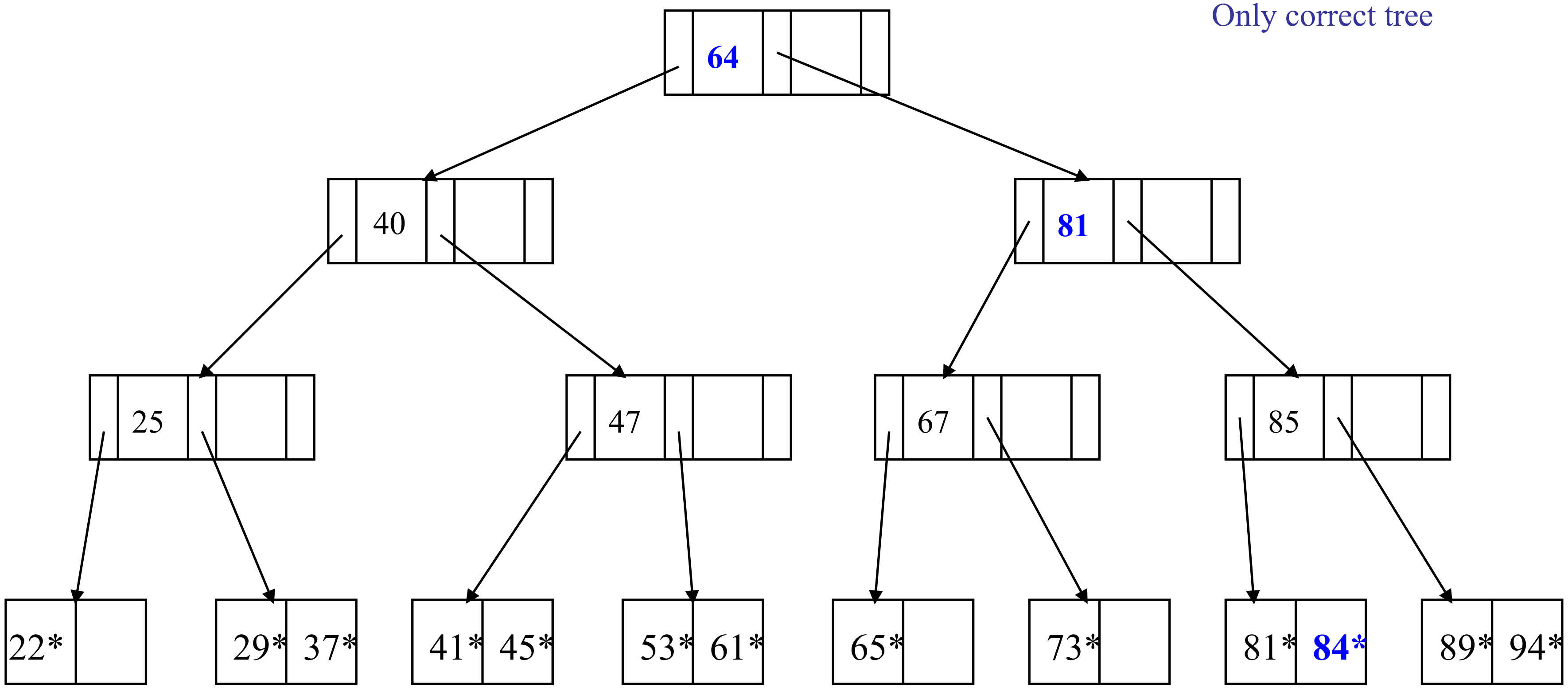
answer on next page



(will be passed up, and split the root by middle value of 64)

Write resulting B+ tree for (b) here

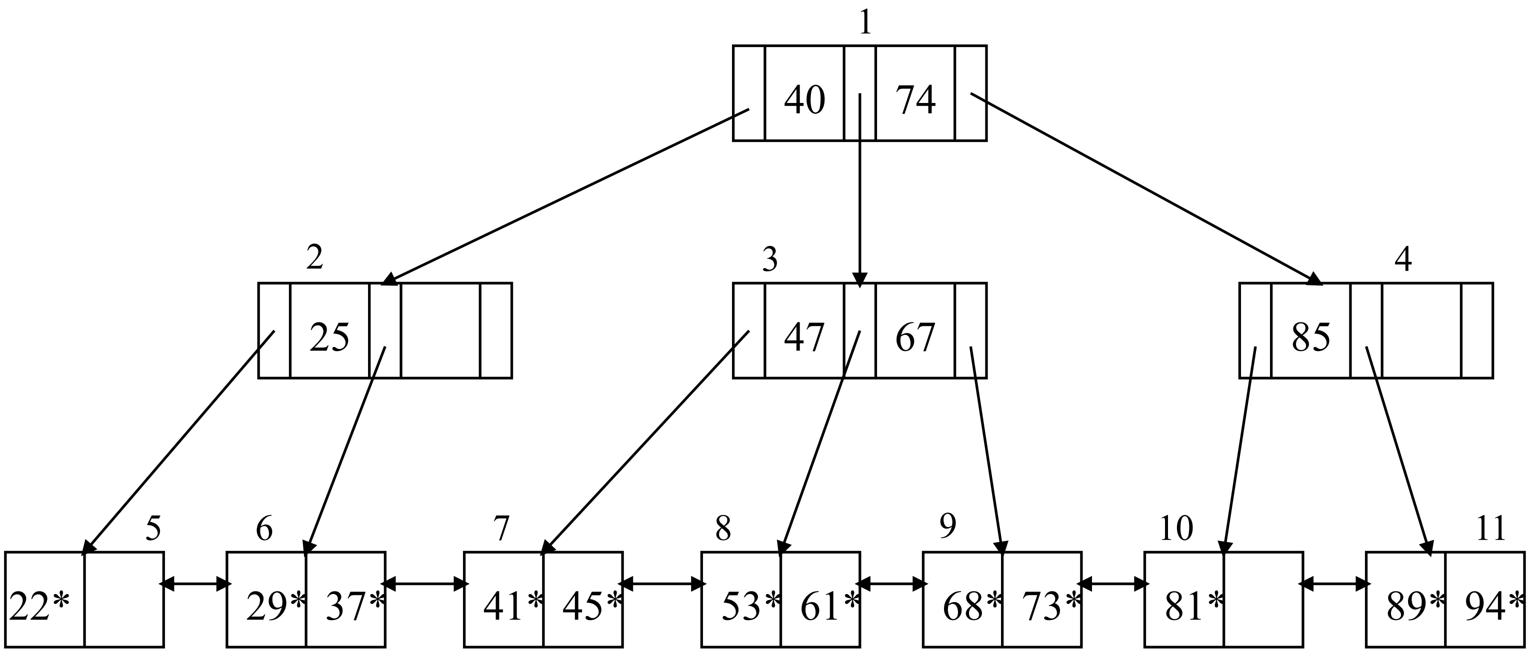
Only correct tree



-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)**

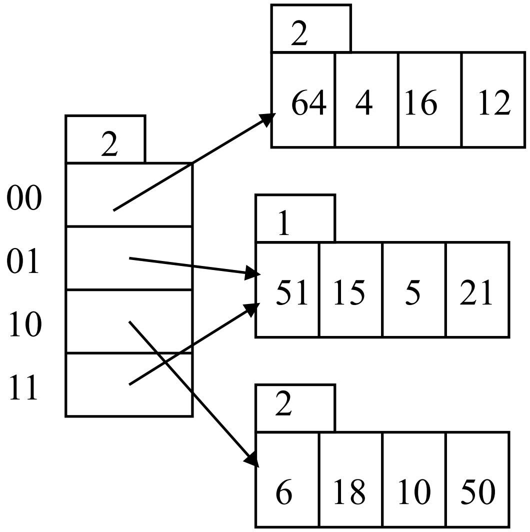
(b) UPDATE T SET T.B = T.B + 100 WHERE T.A = 37 : **S(1), S(2), S(6)**

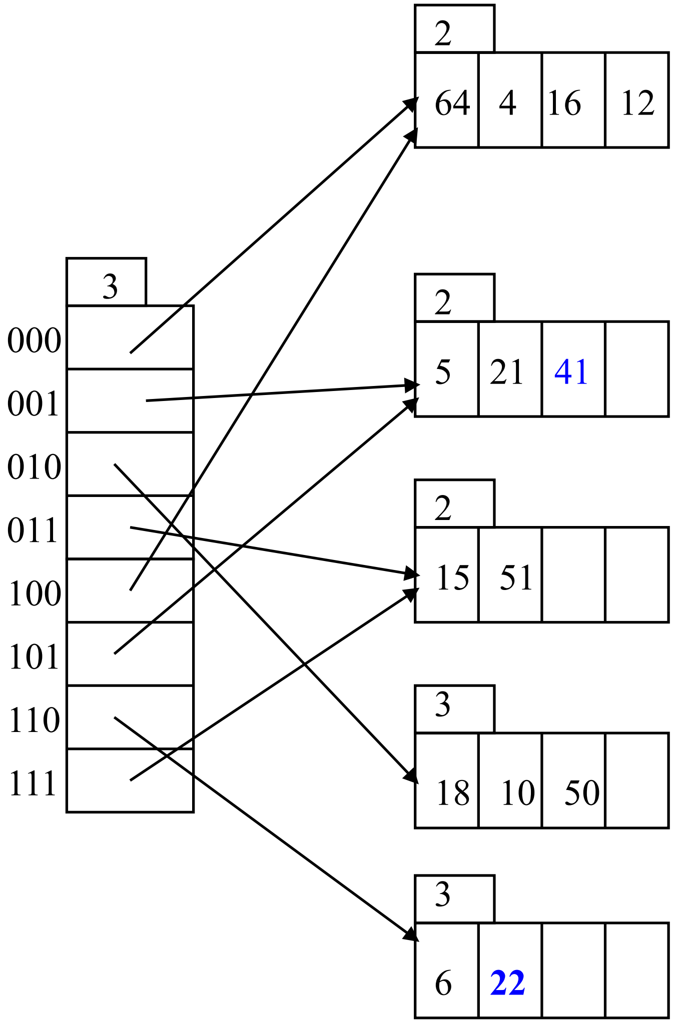
(c) UPDATE T SET T.A = T.A + 5 WHERE T.A = 29 : **S(1), S(2), S(6), X(6)**

**In most cases, probably full credit for selected variations, notably skipping over S(k) straight to X(k)**

(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**

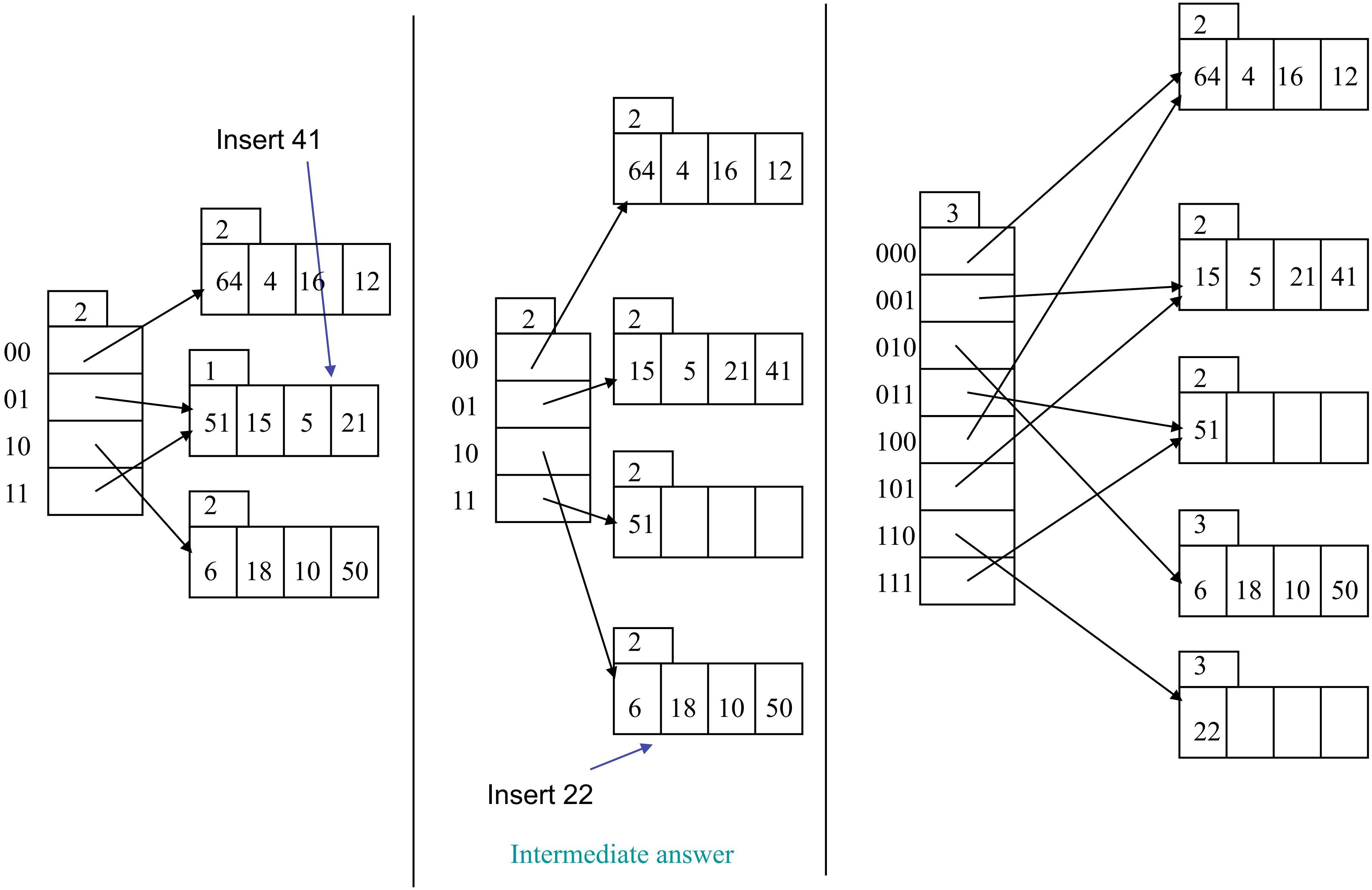
**22. (3 pts)** Consider the extendible hash table to the left. Assume Hash(x) = x. Show the result of inserting the following keys in order:   41, 22





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

Answer here



Insert 41

Insert 22

Intermediate answer