

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

1. (2 pts) Consider CAREFULLY the following table definitions. Remember that these definitions may not reflect the constraints that you think, intuitively, should be true of the database. Nonetheless, you are to assume these definitions, as given, for this question and several subsequent questions.

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

```
CREATE TABLE Product (  
  ProdID Integer,  
  ProdName CHAR[15],  
  Cost Integer,  
  PRIMARY KEY (ProdID))
```

```
CREATE TABLE Account (  
  SSN Integer,  
  AcctNo Integer,  
  Balance Float NOT NULL,  
  PRIMARY KEY (AcctNo),  
  FOREIGN KEY (SSN)  
    REFERENCES Customer  
    ON DELETE CASCADE)
```

```
CREATE TABLE Transaction (  
  ProdID Integer,  
  AcctNo Integer,  
  Date CHAR[6],  
  NumberOfProduct Integer,  
  PRIMARY KEY (AcctNo, ProdID),  
  FOREIGN KEY (AcctNo)  
    REFERENCES Account  
    ON DELETE NO ACTION,  
  FOREIGN KEY (ProdID)  
    REFERENCES Product  
    ON DELETE CASCADE)
```

Explain why the following statement would generate an error (or exception) – BTW: your answer should NOT be “a missing ‘,’”

```
INSERT INTO Transaction  
SELECT DISTINCT ProdID, NULL, NULL, NULL  
FROM Product  
WHERE ProdID NOT IN (SELECT ProdID FROM Transaction)
```

The PK for Transactions includes AcctNo, which can therefore not be NULL, but this INSERT statement would attempt to set AcctNo to NULL

2. (4 pts) Consider CAREFULLY the following table definitions (same as previous problem).

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

```
CREATE TABLE Product (  
  ProdID Integer,  
  ProdName CHAR[15],  
  Cost Integer,  
  PRIMARY KEY (ProdID))
```

```
CREATE TABLE Account (  
  SSN Integer,  
  AcctNo Integer,  
  Balance Float NOT NULL,  
  PRIMARY KEY (AcctNo),  
  FOREIGN KEY (SSN)  
    REFERENCES Customer  
    ON DELETE CASCADE)
```

```
CREATE TABLE Transaction (  
  ProdID Integer,  
  AcctNo Integer,  
  Date CHAR[6],  
  NumberOfProduct Integer,  
  PRIMARY KEY (AcctNo, ProdID),  
  FOREIGN KEY (AcctNo)  
    REFERENCES Account  
    ON DELETE NO ACTION,  
  FOREIGN KEY (ProdID)  
    REFERENCES Product  
    ON DELETE CASCADE)
```

Using these definitions, circle all true statements.

- ☒ (a) A Customer tuple can be associated with more than one Account tuples *multiple Account tuples can be paired with a SSN*
- ☒ (b) A Customer tuple can be associated with zero Account tuples *There need be NO Account tuple that is paired with a given SSN*
- ☐ (c) An Account tuple can be associated with more than one Customer tuples
- ☒ (d) An Account tuple can be associated with zero Customer tuples *SSN can be NULL in Account*
- ☒ (e) An Account tuple can be associated with more than one Transaction tuples *Similar to (a)*
- ☒ (f) An Account tuple can be associated with zero Transaction tuples *Similar to (b)*
- ☐ (g) A Transaction tuple can be associated with more than one Product tuples
- ☐ (h) A Transaction tuple can be associated with zero Product tuples

3. (4 pts) Consider CAREFULLY the following table definitions (same as previous problem).

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

```
CREATE TABLE Product (
  ProdID Integer,
  ProdName CHAR[15],
  Cost Integer,
  PRIMARY KEY (ProdID))
```

```
CREATE TABLE Account (
  SSN Integer,
  AcctNo Integer,
  Balance Float NOT NULL,
  PRIMARY KEY (AcctNo),
  FOREIGN KEY (SSN)
    REFERENCES Customer
    ON DELETE CASCADE)
```

```
CREATE TABLE Transaction (
  ProdID Integer,
  AcctNo Integer,
  Date CHAR[6],
  NumberOfProduct Integer,
  PRIMARY KEY (AcctNo, ProdID),
  FOREIGN KEY (AcctNo)
    REFERENCES Account
    ON DELETE NO ACTION,
  FOREIGN KEY (ProdID)
    REFERENCES Product
    ON DELETE CASCADE)
```

Using these definitions, circle all true statements.

2 pts for one, 3 pts for two, 4 pts for four; -1 for each incorrect circled

- (a) An Account tuple can be associated with a given Product tuple at most once in this DB *(AcctNo, ProdID) is PK of Transaction*
- (b) A Customer tuple can be associated with more than one Product tuple in this DB *See 2(a) option, + Customer join Account join Transaction join Product*
- (c) If a delete command is issued for a tuple of Customer, it will always cause one or more deletes in Account *Not all Customers need participate in Account*
- (d) If a delete command is issued for a tuple of Customer, it will never cause a delete in Transaction *Either Customer has no Account in Transaction, or NO ACTION blocks*
- (e) If a delete command is issued for a tuple of Product, it will always cause a delete in Transaction *Not all Products need participate in Transaction*

4. (5 pts) Write a query in relational algebra that returns the names of all customers with City = 'Nashville' who have ever purchased a product that Costs more than 100 (Cost > 100), together with the ProdName of that product. So, the result will be tuples of the form (Name, ProdName)

$\pi_{\text{Name, ProdName}} (\sigma_{\text{City='Nashville' and Cost > 100}} (\text{Customer} \bowtie \text{Account} \bowtie \text{Transaction} \bowtie \text{Product}))$

3 pts for otherwise correct
SQL query

$\pi_{\text{Name, ProdName}} ((\sigma_{\text{City='Nashville'}} \text{Customer}) \bowtie \text{Account} \bowtie \text{Transaction} \bowtie (\sigma_{\text{Cost > 100}} \text{Product}))$

5. (5 pts) Consider CAREFULLY the following table definitions (same as previous problem).

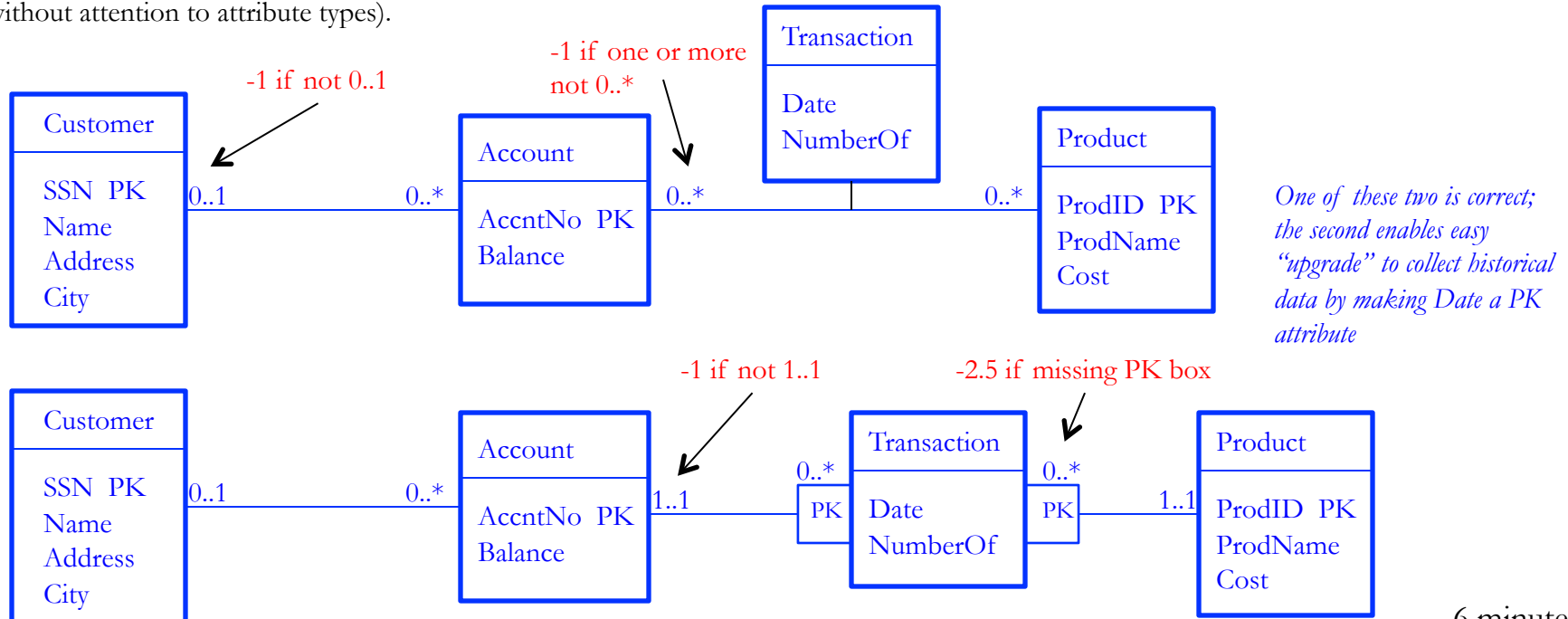
```
CREATE TABLE Customer (
  SSN Integer,
  Name CHAR[25] NOT NULL,
  Address CHAR[25] NOT NULL,
  City CHAR[25] NOT NULL,
  PRIMARY KEY (SSN))
```

```
CREATE TABLE Product (
  ProdID Integer,
  ProdName CHAR[15],
  Cost Integer,
  PRIMARY KEY (ProdID))
```

```
CREATE TABLE Account (
  SSN Integer,
  AcctNo Integer,
  Balance Float NOT NULL,
  PRIMARY KEY (AcctNo),
  FOREIGN KEY (SSN)
    REFERENCES Customer
    ON DELETE CASCADE)
```

```
CREATE TABLE Transaction (
  ProdID Integer,
  AcctNo Integer,
  Date CHAR[6],
  NumberOfProduct Integer,
  PRIMARY KEY (AcctNo, ProdID),
  FOREIGN KEY (AcctNo)
    REFERENCES Account
    ON DELETE NO ACTION,
  FOREIGN KEY (ProdID)
    REFERENCES Product
    ON DELETE CASCADE)
```

Give a UML that is consistent with all the constraints of these table definitions (i.e., the UML would translate to these tables, without attention to attribute types).



6. (5 pts) Consider CAREFULLY the following table definitions (same as previous problem).

*or $SUM(P.Cost * T.NumberOfProduct)$*

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

```
CREATE TABLE Product (  
  ProdID Integer,  
  ProdName CHAR[15],  
  Cost Integer,  
  PRIMARY KEY (ProdID))
```

```
CREATE TABLE Account (  
  SSN Integer,  
  AcctNo Integer,  
  Balance Float NOT NULL,  
  PRIMARY KEY (AcctNo),  
  FOREIGN KEY (SSN)  
    REFERENCES Customer  
    ON DELETE CASCADE)
```

```
CREATE TABLE Transaction (  
  ProdID Integer,  
  AcctNo Integer,  
  Date CHAR[6],  
  NumberOfProduct Integer,  
  PRIMARY KEY (AcctNo, ProdID),  
  FOREIGN KEY (AcctNo)  
    REFERENCES Account  
    ON DELETE NO ACTION,  
  FOREIGN KEY (ProdID)  
    REFERENCES Product  
    ON DELETE CASCADE)
```

Define a VIEW called AllPurchases with a schema that contains 5 attributes: a Customer Name and Address; ProdID and ProdName of a Product purchased by the Customer; and the sum of Cost for that Product, by that Customer (i.e., the sum of Product Cost multiplied by the Transaction NumberOfProduct). A final constraint is that the view should only list entries (Name, Address, ProdID, ProdName, Total) in cases where the sum of NumberOfProduct exceeds 100. Do not use JOIN keywords.

-0.5 if SUM missing

CREATE VIEW AllPurchases AS

SELECT C.Name, C.Address, P.ProdID, ProdName, P.Cost * SUM(T.NumberOfProduct) *or $SUM(P.Cost * T.NumberOfProduct)$*

FROM Customer C, Account A, Product P, Transaction T

WHERE C.SSN = A.SSN AND A.AcctNo = T.AcctNo AND T.ProdID = P.ProdID

GROUP BY C.SSN, C.Name, C.Address, P.ProdID, P.ProdName, P.Cost

HAVING SUM(T.NumberOfProduct) > 100

-0.5 for not including C.SSN in GROUP BY (or ProdID)

-1 if missing HAVING (or incorrect -0.5) HAVING clause

Include all attributes that are used in SELECT and that don't appear in an aggregate operator (SQL standard); include C.SSN because C.Name and C.Address not identified as key of Customer. Should include C.SSN and P.ProdID at minimum in GROUP BY

While there can be only one pairing of an Account and a Product, there can be multiple pairings of a Customer and a Product (through multiple Accounts)

6 minutes

7. (5 pts) Consider CAREFULLY the following table definitions (same as previous problem).

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

```
CREATE TABLE Product (  
  ProdID Integer,  
  ProdName CHAR[15],  
  Cost Integer,  
  PRIMARY KEY (ProdID))
```

```
CREATE TABLE Account (  
  SSN Integer,  
  AccntNo Integer,  
  Balance Float NOT NULL,  
  PRIMARY KEY (AccntNo),  
  FOREIGN KEY (SSN)  
    REFERENCES Customer  
    ON DELETE CASCADE)
```

```
CREATE TABLE Transaction (  
  ProdID Integer,  
  AccntNo Integer,  
  Date CHAR[6],  
  NumberOfProduct Integer,  
  PRIMARY KEY (AccntNo, ProdID),  
  FOREIGN KEY (AccntNo)  
    REFERENCES Account  
    ON DELETE NO ACTION,  
  FOREIGN KEY (ProdID)  
    REFERENCES Product  
    ON DELETE CASCADE)
```

Write a CREATE TRIGGER statement that deletes an Account tuple when the only Transaction tuple involving that Account is deleted

```
CREATE TRIGGER DeleteAccountWithNoTransactions  
AFTER DELETE on Transaction  
WHEN NOT EXISTS (SELECT *  
                  FROM Transaction T  
                  WHERE T.AccntNo = old.AccntNo)
```

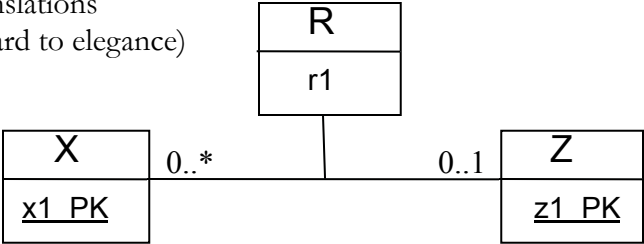
Including "FOR EACH ROW" is optional in SQLite, which I expect most of you will assume, because all triggers in SQLite are row level triggers

Or WHEN old.AccntNo NOT IN (SELECT AccntNo FROM Transactions)

```
BEGIN  
DELETE FROM Account A WHERE A.AccntNo = old.AccntNo;  
END;
```

```
CREATE TRIGGER DeleteAccountWithNoTransactions  
AFTER DELETE on Transaction  
BEGIN  
DELETE FROM Account A WHERE A.AccntNo = old.AccntNo AND  
                           A.AccntNo NOT IN (SELECT T.AccntNo FROM Transaction T);  
END;
```

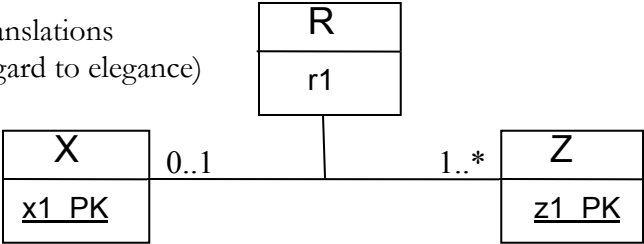
8. (5 pts) Consider the UML fragment to the right and identify (circle) all equivalent table translations (i.e., those translations that faithfully enforce the constraints implied by the UML without regard to elegance) from those given below. You might receive partial credit for a brief explanation of your choices. Assume that UNIQUE(y) implies that y NOT NULL, but not vice versa. PK stands for PRIMARY KEY. FK stands for FOREIGN KEY.



3 points for one, 5 points for two

(a)	(b)	(c)	(d)	(e)
<pre> 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)) </pre>	<pre> 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)) </pre>	<pre> CREATE TABLE XR (x1, r1, z1, PK(x1), FK (z1) refs Z) CREATE TABLE Z (z1, PK(z1)) </pre>	<pre> CREATE TABLE XR (x1, r1, z1 NOT NULL, PK(x1), FK (z1) refs Z) CREATE TABLE Z (z1, PK(z1)) </pre>	<pre> CREATE TABLE XR (x1, r1, z1, PK(x1), UNIQUE(z1), FK (z1) refs Z) CREATE TABLE Z (z1, PK(z1)) </pre>
-1 points			-1 points	-2 points
			(f) None of the above	
			0 points if circled, with or without other choices	

9. (5 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.

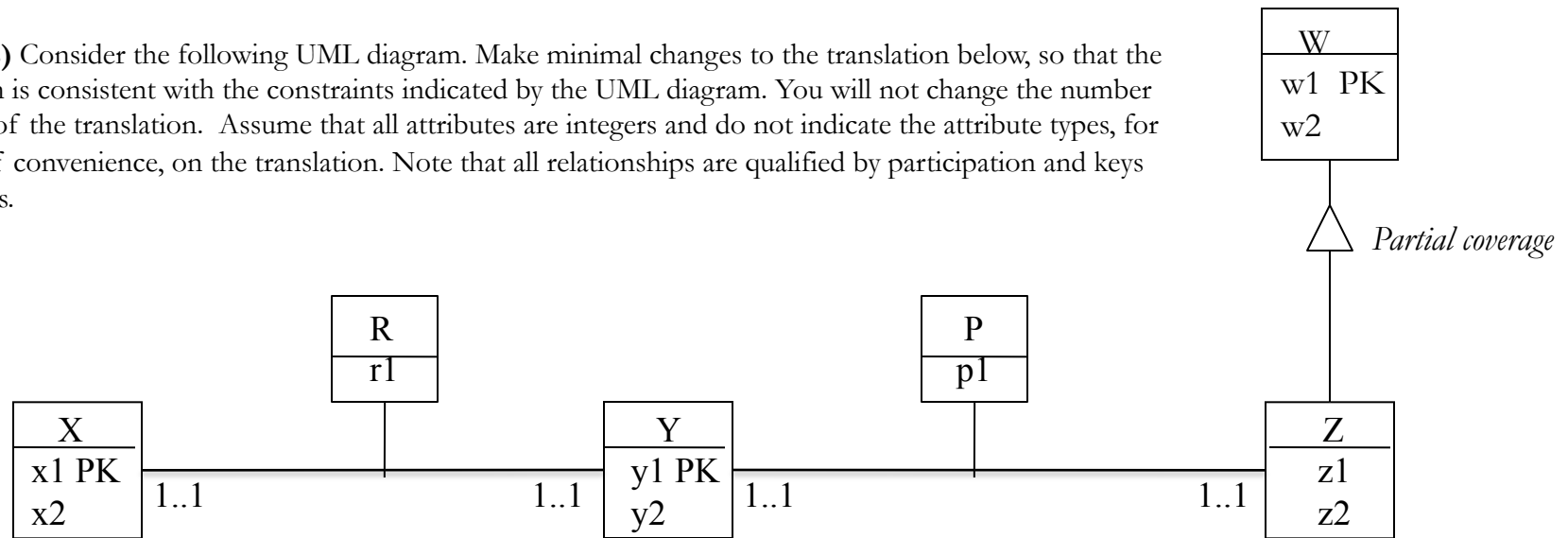


3 points for one, 5 points for two

(a)	(b)	(c)	(d)	(e)
<p>CREATE TABLE X (x1, PK (x1))</p> <p>CREATE TABLE R (x1, r1, z1, PK(x1, z1), FK (z1) refs Z, FK (x1) refs X)</p> <p>CREATE TABLE Z (z1, PK (z1))</p> <p>CREATE ASSERTION XparticipatesZ CHECK (NOT EXISTS (SELECT * FROM X WHERE X.x1 NOT IN (SELECT R.x1 FROM R)))</p>	<p>CREATE TABLE X (x1, PK (x1))</p> <p>CREATE TABLE Z (x1, r1, z1 PK(z1), FK (x1) refs X)</p> <p>-2 points (an X need not participate with a Z)</p>	<p>CREATE TABLE X (x1, PK (x1))</p> <p>CREATE TABLE Z (x1, r1, z1, PK(z1), FK (x1) refs X)</p> <p>CREATE ASSERTION XparticipatesZ CHECK (NOT EXISTS (SELECT * FROM X WHERE X.x1 NOT IN (SELECT Z.x1 FROM Z)))</p>	<p>CREATE TABLE X (x1, PK (x1))</p> <p>CREATE TABLE R (x1 NOT NULL, r1, z1, PK(z1), FK (z1) refs Z, FK (x1) refs X)</p> <p>CREATE TABLE Z (z1, PK (z1))</p> <p>CREATE ASSERTION XparticipatesZ CHECK (NOT EXISTS (SELECT * FROM X WHERE X.x1 NOT IN (SELECT R.x1 FROM R)))</p>	<p>(e) None of the others</p> <p>0 points if circled, with or without other choices</p>

-1 points (a Z can participate
with more than one X, through R)

10. (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) or some other rearrangement
that makes EACH of W1, Y1, and X1 a key (primary or
candidate)

1 pt for UNIQUE(W1, Y1)

11. (5 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))
```

3 pts for one,
4 pts for two,
5 pts for three

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) 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) 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 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 - 2pts

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

(e) None of the others

0 total

*Look particularly carefully for the small difference between
(c) and (d)*