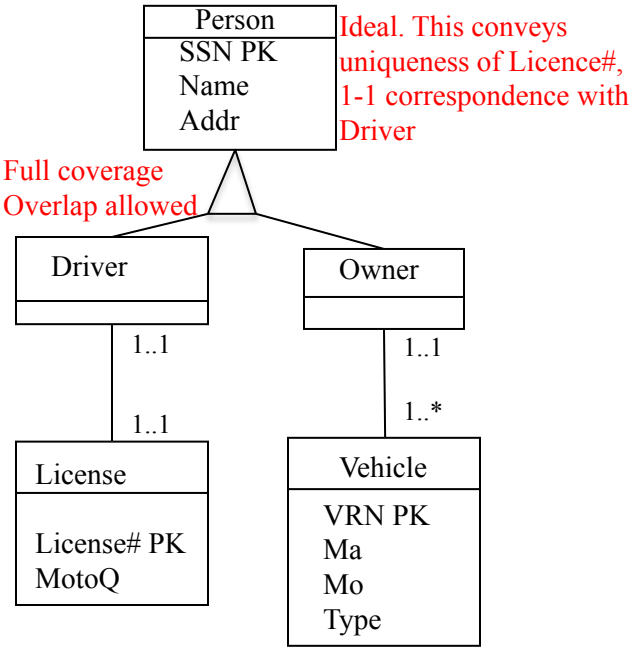
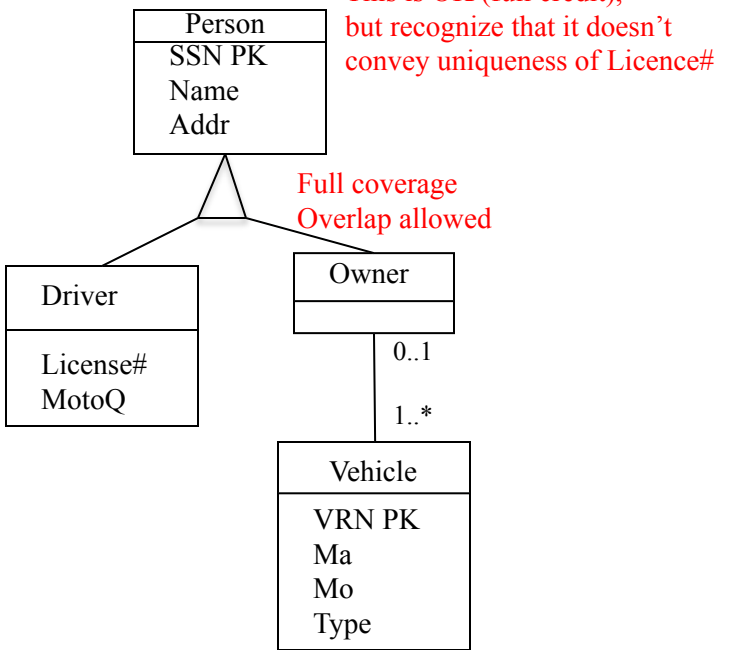
Midterm (CS 265) Name:_____KEY_____

I will not notes use notes, other people's exams, or other sources in this exam (please sign): _____

1.(5 points) Give a UML diagram that captures the following information for a simplified DMV/TDOT-like database. Your UML should include a hierarchy of subclasses that are *complete coverage* and *overlap allowed*.

a) A Person can be a licensed vehicle driver and/or a vehicle owner. A person is uniquely identified by SSN, and is also described by name and address. Only persons that are licensed drivers and/or owners are to be recorded in the DB (a driver need not own a vehicle, and an owner need not be a licensed driver).
b) Each vehicle is identified uniquely by vehicle registration number (VRN), with other attributes: make, model, and type (e.g., motorcycle, car, truck).
c) Each vehicle is owned by AT MOST one person (i.e., owner).
d) A driver has exactly one driver's license
e) Each license is associated with exactly one driver, and has an attribute license number that uniquely identifies the license and an (Boolean) attribute MotorcycleQualified (MotoQ).





2. (3 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)

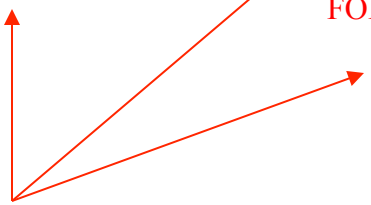
Products (*ProductId*: integer, *ProductName*: string, *cost*: integer)

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 ensure that a *product* can only be deleted if it does not participate in any transaction (as identified by *ProductId*). 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
)
```

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



```
CREATE TABLE Products (
  ProductID Integer,
  ProductName CHAR[15] NOT NULL,
  Cost Integer NOT NULL,
  PRIMARY KEY (ProductId)
)
```

-1 FOR EACH if not completely right (but forgive minor syntax errors) or missing

3. (4 points) Given the following relation with the listed FDs: R(A, B, C, D, E, F, G, H)

$$A \rightarrow B,\quad B \rightarrow C,\quad C \rightarrow A,\quad AC \rightarrow E,\quad B \rightarrow D,\quad C \rightarrow D,\quad F \rightarrow G$$

Circle all the answers that are TRUE:

2 for first one right, 1 for additional righ pickts
-1 for each circled that is wrong

a) {A, G, H} is a key

BUT not minimal; probably wrong next time (won't take off for (b), but no added point)

b) {A, B, F, G, H} is a key

c) AC → E can be replaced with C → E

d) {C, F, H} is a key

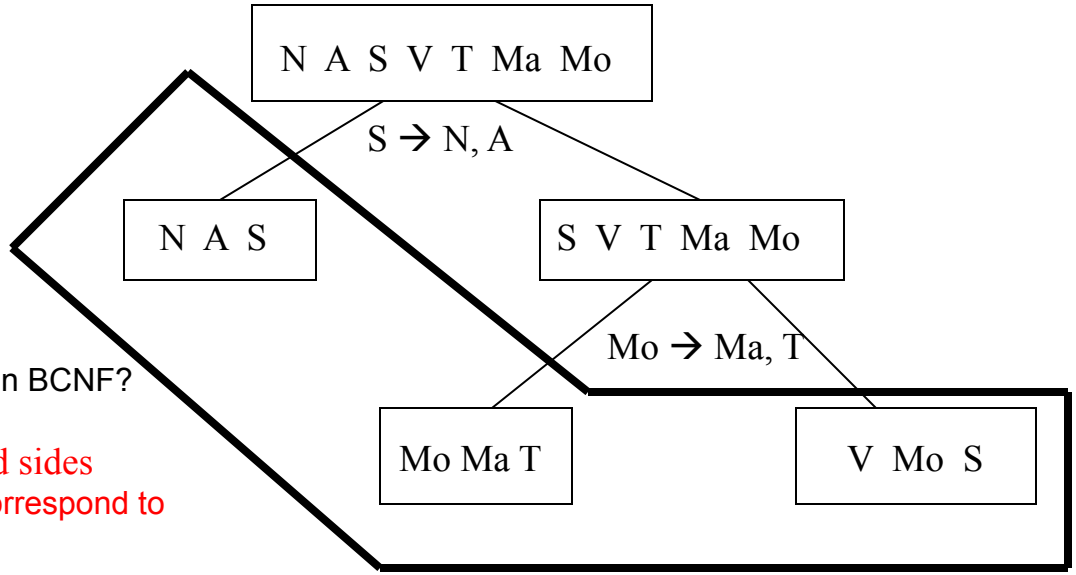
e) F → G is informationally redundant (assuming all remaining FDs)

f) A → B is informationally redundant (assuming all remaining FDs)

g) C → D is informationally redundant (assuming all remaining FDs)

4. (2 pt) Consider the following FDs and the lossless decomposition into three tables (relations) based on them below.

S → N, A
V → Mo, S
Mo → Ma, T



For each of the tables in the decomposition, indicate whether it is in BCNF?

Explain. Yes, all in BCNF; left hand sides of all FDs assignable to each table correspond to keys of the tables

5. Consider the relation R with 5 attributes, R = ABCDE. You are given the following functional dependencies: F = {ED→ A, A→BD, ED → B}. BE CAREFUL: errors may cascade in this problem and they are meant too. Check your work carefully.

a) (2 pts) Give a minimal set of functional dependencies that is equivalent to F (simplify FD left-hand sides and eliminate FDs as appropriate)

ED→A, A→BD

1 pt for each; should be exact for each, excepting order of attributes on LHS and on RHS; A→BD might be broken up Into A→B and A→ D

b) (2 pts) List all keys of R.

ACE, DCE

1 pt for each

6. Assume the following relational schemas (underlined variables in each each schema make up its primary key).

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

Accounts (*SSN*: integer, *AccntNo*: integer)

Transactions (*AccntNo*: integer, *ProductId*: integer, *date*: string, *quantity*: integer) // *quantity* is the number of ProductId purchased on given transaction

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

Write SQL queries for the following. You may NOT use nested queries and you may NOT use any JOIN keyword constructs.

a) (1 pt) Write an SQL query that lists the names of all customers living in Nashville (i.e., city = 'Nashville').

```
SELECT C.Name FROM Customers C WHERE C.city = 'Nashville'
```

b) (1 pt) Write an SQL query that lists the names and account numbers of all customers (regardless of city).

```
SELECT C.Name, A.AcctNo
FROM Customers C, Accounts A
WHERE C.SSN = A.SSN
```

6. Continued -- Using the same relational schemas, write SQL queries for the following

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

Accounts (*SSN*: integer, *AccntNo*: integer)

Transactions (*AccntNo*: integer, *ProductId*: integer, *date*: string, *quantity*: integer) // *quantity* is the number of ProductId purchased on given transaction

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

c) (3 pts) Compute and list the accounts (as *AccntNo*) and purchases (as *ProductId, ProductName, quantity*) on a given date (*date = X*) of all customers living in Nashville. If a customer bought the same product in more than one transaction on the given day, the query would list these in separate rows of the result.

```
SELECT A.AcctNo, P.ProductId, P.ProductName, T.quantity
FROM Customers C, Accounts A, Transactions T, Products P
WHERE C.city = 'Nashville' AND C.SSN = A.SSN AND A.AcctNo = T.AcctNo AND T.date = X
        AND T.ProductId = P.ProductId
```

d) (1 pts) List the total number (sum) of all quantities for the product identified by ProductId = X, regardless of AccntNo, that were purchased on date = Y.

```
SELECT SUM(T.quantity)
FROM Transactions T
WHERE T.ProductId = X AND T.date = Y
```

e) (2 pts) For EACH transaction date and ProductId, list the date, ProductId, and the total number (sum) of all the product's quantities purchased on that date.

```
SELECT T.ProductId, T.date, SUM(T.quantity)
FROM Transactions T
GROUP BY T.ProductId, T.date
```

6. Continued -- Using the same relational schemas, write SQL queries for the following

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

Accounts (*SSN*: integer, *AccntNo*: integer)

Transactions (*AccntNo*: integer, *ProductId*: integer, *date*: string, *quantity*: integer) // *quantity* is the number of ProductId purchased on given transaction

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

f) (3 pts) For each product, list the item's ProductId, ProductName, and total (sum) of all item quantities purchased by Nashville customers .

```
SELECT P.ProductId, P.ProductName, SUM(T.quantity)
FROM Customers C, Accounts A, Transactions T, Products P
WHERE C.city = 'Nashville' AND C.SSN = A.SSN AND A.AcctNo = T.AcctNo AND
        T.ProductId = P.ProductId
GROUP BY P.ProductId, P.ProductName
```

Won't take off this time for missing ProductName

g) (2 pts) For EACH item, list the item's ProductId, ProductName, and total (sum) of all item quantities purchased by Nashville customers, BUT ONLY for those product's with cost > 50 and having a total purchased quantity of greater than 100.

```
SELECT P.ProductId, P.ProductName, SUM(T.quantity)
FROM Customers C, Accounts A, Transactions T, Products P
WHERE C.city = 'Nashville' AND C.SSN = A.SSN AND A.AcctNo = T.AcctNo AND
        T.ProductId = P.ProductId AND P.cost > 50
GROUP BY P.ProductId, P.ProductName
HAVING SUM(T.quantity) > 100
```

7. Consider the following relational schema (underlined variables in each schema make up its primary key)

Suppliers (*sid*: integer, *sname*: string, *address*: string, *city*: string)

Catalog (*sid*: integer, *pid*: integer, cost: real)

Parts (*pid*: integer, *pname*: string, *color*: string)

Write RELATIONAL ALGEBRA queries for the following

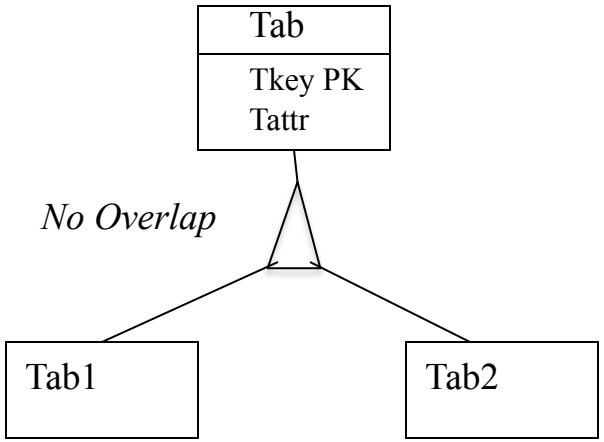
a) (2 points) Find the *snames* of all Suppliers that supply any **green** part.

Project(sname) (select(color=Green) (Suppliers njoin Catalog njoin Parts)) njoin means natural join

b) (3 points) Find the *sids* of Suppliers that supply any **green** part and are in the city of **Nashville**.

Project(sid) (select(color=Green and city=Nashville) (Suppliers njoin Catalog njoin Parts))

8. (3 pts) Circle all options that would correctly enforce the No Overlap (aka Disjoint) constraint between Tab1 and Tab2 subclasses of Tab in an SQL translation of the following UML fragment.



-1 for each other option than (a) and (b) (again, a negative score will simply be counted Zero/0) ; if (f) made on exam of someone leaving before announcement of typo, then count it as OK, in place of or in addition to option (b)

1.5 points for each

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

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

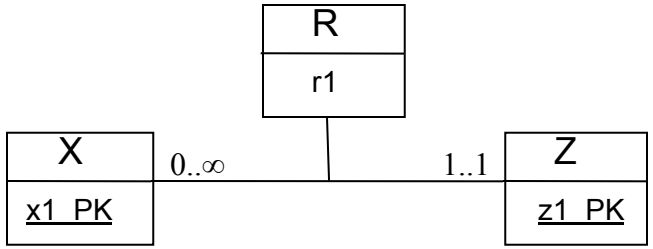
c) CREATE ASSERTION NoOverlapBetweenTab1AndTab2
CHECK ((SELECT COUNT (Tab1.Tkey) FROM Tab1) = (SELECT COUNT (Tab2.Tkey) FROM Tab2))

d) CREATE ASSERTION NoOverlapBetweenTab1AndTab2
CHECK ((SELECT COUNT (Tab1.Tattr) FROM Tab1) = (SELECT COUNT (Tab2.Tattr) FROM Tab2))

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

f) None of the above 0 points for this

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 of 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) 2 points

```
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)
)
```

(B)

```
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)
)
```

-2 points

(C)

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

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

-1 points

(D) 3 points

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

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

(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

-5 points

10. (2 pts) Consider the following table definitions (attribute types omitted).

```
CREATE TABLE Readings (
   Kilowatts, Date, Time, LocationId,
  PRIMARY KEY (Date, Time, LocationId),
  FOREIGN KEY (Date, Time) REFERENCES TimeStamps,
  FOREIGN KEY (LocationId) REFERENCES  Locations )

CREATE TABLE Locations (
    LocationId, Occupancy, …
    PRIMARY KEY (LocationId) )

CREATE TABLE LocatedIn (
    IPAddress, LocationId, …
    PRIMARY KEY (IPAddress),
    FOREIGN KEY (LocationId) REFERENCES Locations)
```

Write a query that returns the *average* Kilowatts over all readings for the LocationIds associated with EACH IPAddress on a given Date (Date = D, where D is a parameter). In addition to the averages, the result should list the IPAddress and LocationId for each reported average. Moreover, the query should only return the averages for IRAddress/LocationId groups for which the minimum Kilowatt reading is greater than 0.5 on date D.

Complete the following query so that properly implements the specification.

```
SELECT L.IPAddress, L.LocationId, AVG(R.Kilowatts)
FROM Readings R, LocatedIn L
WHERE L.LocationId = R.LocationId AND R.Date = D
GROUP BY L.IPAddress, L.LocationId
HAVING MIN(R.Kilowatts) > 0.5
```

1 point for correct GROUP BY (adding LocId)
1 point for correct HAVING