Exam 2 CS x265 Spring 2017 Name: _____KEY_____

To request a regrade or to otherwise appeal your score, submit a one paragraph explanation of why you think the grade was "in error" and submit your explanation/request together with your exam to Doug after class or in office hours.

The lowest possible on any problem is 0 (regardless of how many negative points might be assigned by rubric)

Exam 2 CS x265 Spring 2017 Name: _____KEY_____

I will not use notes, other exams, or any source other than my own brain on this exam: _____(please sign)

1. (2 pts) You have two relations R (C, F, G) and P (B, C, D, F, H). Using only relational algebra select (σ)and cross product (X) operators, express the natural join of R and P.

$\sigma_{R.C=P.C \text{ AND } R.F=P.F}(R \times P)$ OR $\sigma_{R.F=P.F}\ \sigma_{R.C=P.C}(R \times P)$

-1 for any missing major component like a σ or a X, or a confition on one of these. Forgive what you think is an obvious typo or minor syntax error

2. (3 pts) Consider the following four relational schema, with the key for each relation underlined

Customer (*SSN*: integer, *name*: string, *address*: string, *city*: string)
Account (*SSN*: integer, *AccntNo*: integer, *balance*: real)
Transaction (*AccntNo*: integer, *ProdId*: integer, *date*: string, *quantity*: integer)
Product (*ProdId*: integer, *ProdName*: string, *cost*: real)

Rewrite the following query using only JOIN-keyword operations for joining (you may assume SQLite)

See next page for possible answers

```
SELECT A.AcctNo, P.ProdId, P.ProdName, T.quantity
FROM Customer C, Account A, Transaction T, Product P
WHERE C.city = 'Nashville' AND T.date = X
        AND C.SSN = A.SSN
        AND A.AcctNo = T.AcctNo
        AND T.ProdId = P.ProdId
```

6 minutes

```
SELECT A.AcctNo, P.ProdId, P.ProdName, T.quantity
FROM Customer C, Account A, Transaction T, Product P
WHERE C.city = 'Nashville' AND T.date = X
        AND C.SSN = A.SSN
        AND A.AcctNo = T.AcctNo
        AND T.ProdId = P.ProdId
```

```
SELECT A.AcctNo, P.ProdId, P.ProdName, T.quantity
FROM Customer C NATURAL JOIN Account A
        NATURAL JOIN Transaction T
        NATURAL JOIN Product P
WHERE C.city = 'Nashville' AND T.date = X
        -AND C.SSN = A.SSN
        AND A.AcctNo = T.AcctNo
        AND T.ProdId = P.ProdId
```

```
SELECT A.AcctNo, P.ProdId, P.ProdName, T.quantity
FROM (((Customer C INNER JOIN Account A  ON C.SSN = A.SSN)
        INNER JOIN Transaction T ON A.AcctNo = T.AcctNo
        INNER JOIN Product P ON T.ProdId = P.ProdId))
WHERE C.city = 'Nashville' AND T.date = X
```

```
SELECT A.AcctNo, P.ProdId, P.ProdName, T.quantity
FROM (((Customer C JOIN Account A  ON C.SSN = A.SSN)
        JOIN Transaction T ON A.AcctNo = T.AcctNo
        JOIN Product P ON T.ProdId = P.ProdId))
WHERE C.city = 'Nashville' AND T.date = X
```

```
SELECT A.AcctNo, P.ProdId, P.ProdName, T.quantity
FROM (((Customer C JOIN Account A  USING (SSN)
        JOIN Transaction T USING(AcctNo))
        JOIN Product P USING(ProdId))
WHERE C.city = 'Nashville' AND T.date = X
```

and still other variations, which can include not including C, A, T, P, but rather prefacing by table name (Customer, Account, Transaction, Product) as needed, instead.

Best to parenthesize in case of these options, but no points off if conditions ON or USING are placed adjacent to their corresponding JOIN operations

-1 if OITER JOIN used (changes the rows returned in result)
-1 missing one or both WHERE conditions
-2 for two-three missing JOIN keywords (assume typo if only one)
-2 if JOIN used (without NATURAL prefix), and ON or USING is not used
-3 (i.e., 0 score) if JOIN keyword not used as all

JOIN alone implies INNER JOIN;
Can't use USING and ON in same clause (e.g., for single join, but could see examples of intermixing ON and USING in this case (but no good reason to since all equality joins)

6 minutes

3. (4 pts) Consider the same relational schemas as problem 2 (without quantity for Transaction).

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

Account (*SSN*: integer, *AccntNo*: integer, *balance*: real)

Transaction (*AccntNo*: integer, *ProdId*: integer, *date*: string)

Product (*ProdId*: integer, *ProdName*: string, *cost*: integer)

Consider the CREATE TABLE statements that implement the relations to the left. SSN in Account refers to an SSN in Customer. AcctNo in Transaction refers to an AccntNo in Account. ProdId in Transaction refers to a ProdId in Product. **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 all actions in the appropriate definitions.

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

2 pts for one FK declaration properly done;
3 pts for two, 4 pts for all three.
if UPDATE included, it would probably be same actions as for DELETE, but don't grade UPDATE actions

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

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

or RESTRICT instead of NO ACTION

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

or could modify attribute declarations themselves to include FK actions, such as

AccntNo Integer REFERENCES Account ON DELETE NO ACTION

-2 FOR any other change, most notably the inclusion of FK constraints in Customer or Product

6 minutes

4. (4 pts) Consider the following table definitions (attribute types omitted).

```
CREATE TABLE Reading (
  Kilowatts, Date, Time, LocationId,
  PRIMARY KEY (Date, Time, LocationId),
  FOREIGN KEY (LocationId) REFERENCES  Location )

CREATE TABLE Location (
    LocationId, Occupancy, …
    PRIMARY KEY (LocationId) )
```

We want a query that returns the *average* Kilowatts over all readings for each Location on a given Date (Date = D, where D is a parameter). In addition to the averages, the result should list the LocationId and the locations' occupancy for each reported average. Moreover, the query should only return the averages for LocationIds for which the minimum individual Kilowatt reading at the location is greater than 0.5 on date D.

Complete the following query so that properly implements the specification.

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

2 pts for correct GROUP BY (with L.LocationId or R.LocationId and L.Occupancy. Only 1.5 if Occupancy missing

2 pts for correct HAVING as given

2 minutes

-2 or more for any other additions

5. (3 pts) Consider the two tables, SV and VM below and on the left. Each table has two attributes.

SV

S, V
S1 V4
S1 V5
S2 V6

VM

V , M
V1 M1
V2 M2
V3 M1
V4 M3
V5 M4
V6 M2
V7 M3

S	V	M
NULL	V1	M1
NULL	V2	M2
NULL	V3	M1
S1	V4	M3
S1	V5	M4
S2	V6	M2
NULL	V7	M3

Result of query

Circle all queries below that would yield the result on the right.

+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

(e) SV NATURAL FULL OUTER JOIN Vmo

(f) None of the above

-1.0 for each

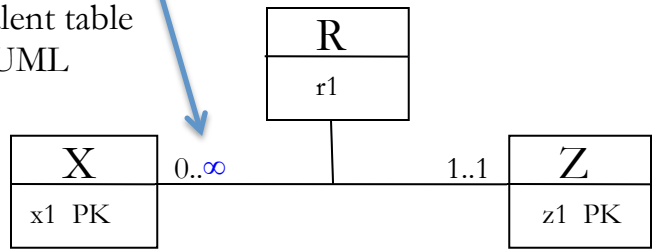
0 total

If you picked (c), (d), and (e) only, I figured you just got your directions wrong and counted correct, but not next time

5 minutes

Should have been 0..* . Contact if confused

6. (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.



4 pts for one choice, 5 pts for both

(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)
)
```

Not as natural, but enforces constraints

(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

This enforces 0..1 from XR to Z, not 1..1

(C)

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

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

-1 points

This enforces 0..1 from XR to Z, not 1..1

(D)

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

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

I would expect almost all to get this one

(E)

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

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

-2 points

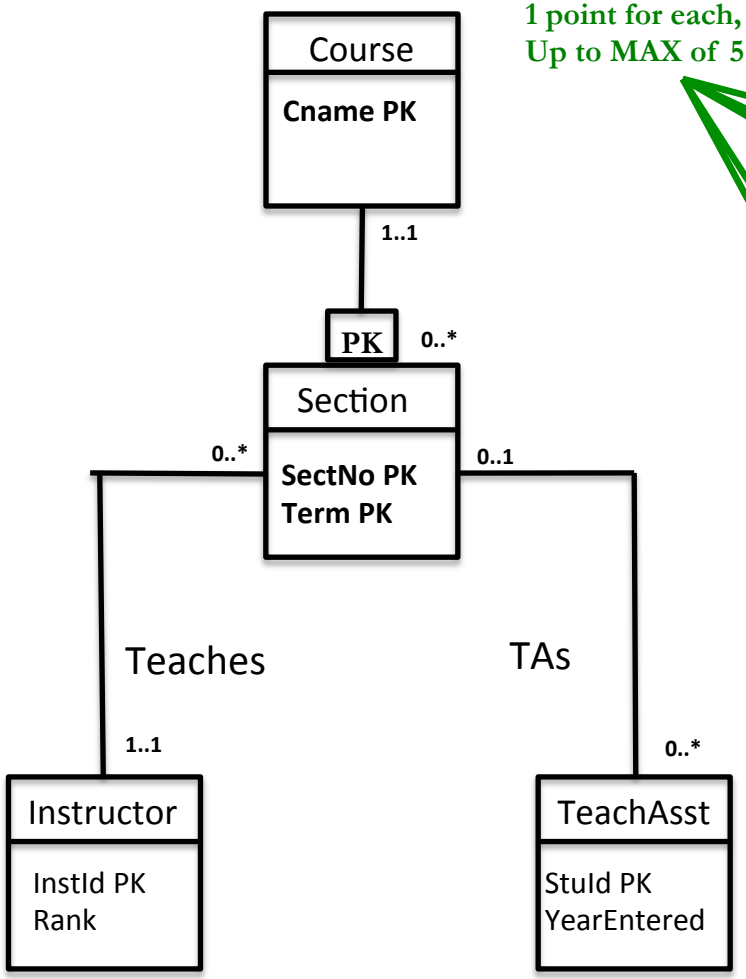
UML does not require each X to be associated with unique Z

(F) None of the above

-5 points

6 minutes

7. (5 pts) Consider the following UML snippet. Debug the definitions of **Section** and **TeachAsst** that are given, so that they conform to the constraints of the UML specification, assuming that the other tables (i.e., Course and Instructor) have been correctly translated. You may add text and/or strike out text in the current definitions. Be neat and clear. Note that we are not interested in attribute types in this problem.



1 point for each,
Up to MAX of 5 points

```
CREATE TABLE Section (
        SectNo,
        Term,
        Cname,
        InstID, NOT NULL
        PRIMARY KEY (SectNo, Term, Cname),
        FOREIGN KEY (InstID) REFERENCES Instructor,
        FOREIGN KEY (Cname) REFERENCES Course
)
```

including the PK of Instructor is the preferred way of enforcing 1..1, but without NOT NULL, 0..1 is enforced, not 1..1

Make link from Section to Course and Instructor explicit with FK constraints

```
CREATE TABLE TeachAsst (
        SectNo NOT NULL,
        Term NOT NULL,
        Cname NOT NULL,
        StuID,
        YearEntered,
        PRIMARY KEY (SectNo, Term, Cname, StuID),
        FOREIGN KEY (SectNo, Term, Cname)
                REFERENCES Section
)
```

By declaring NOT NULL, 1..1 is enforced, not 0..1

Only one of these NOT NULL needs to be crossed out to make technically correct

The PK given initially allows a TeachAsst (StuID) to be associated with multiple Sections

No points should be taken off if NOT NULLs were added to attributes that were primary key attributes, though NOT NULL is implied already

-1 for additional changes not given above, except as noted otherwise

5 minutes

8. (4 pts) Circle all options that would correctly enforce the 1..* cardinality constraint that Tab1 participate at least once with a record of Tab2 in an SQL translation of the following UML fragment.

+3 points for one correct option (a, b); +4 points for both correct options

a) CREATE ASSERTION Tab1INTab2
CHECK (NOT EXISTS (SELECT * FROM Tab1 WHERE Tab1.Key1 NOT IN (SELECT Tab2.Key1 FROM Tab2)))

b) CREATE ASSERTION Tab1INTab2
CHECK (NOT EXISTS (SELECT Tab1.Key1 FROM Tab1 EXCEPT SELECT Tab2.Key1 FROM Tab2))

c) CREATE ASSERTION Tab1INTab2 If even one Tab1.Key1 is represented in Tab2, this evaluates to TRUE
CHECK (EXISTS (SELECT * FROM Tab1 WHERE Tab1.Key1 IN (SELECT Tab2.Key1 FROM Tab2)))

d) CREATE ASSERTION Tab1INTab2 Evaluates to TRUE if no Key1s in common between Tab1 and Tab2
CHECK (NOT EXISTS (SELECT Tab1.Key1 FROM Tab1 INTERSECT SELECT Tab2.Key1 FROM Tab2))

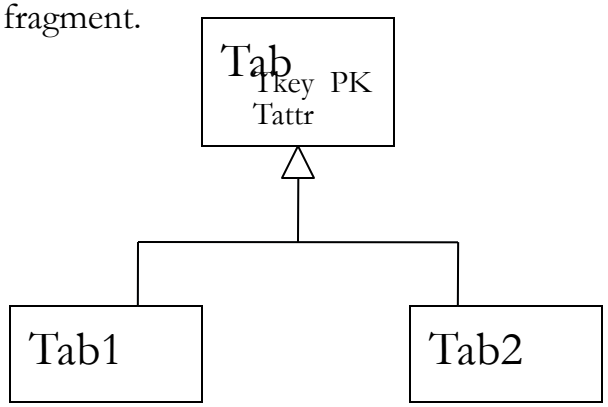
e) CREATE ASSERTION Tab1INTab2 If even one Tab1.Key1 is represented in Tab2, this evaluates to TRUE
CHECK (EXISTS (SELECT Tab1.Key1 FROM Tab1 IN (SELECT Tab2.Key1 FROM Tab2)))
Pair of parens left out, but wrong anyway

f) None of the above **0 if this option circled**

-1 for one incorrectly circled option (from c, d, e);
-3 for two incorrectly circled options;
-5 for 3 incorrectly circled option

6 minutes

9. (3 pts) Circle all options that would correctly enforce the Disjoint constraint (i.e., no overlap allowed) between Tab1 and Tab2 in an SQL translation of this UML fragment.



-1 for each option (a) or (c)

+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

6 minutes

10. (5 pts) Assume that you have a DB with a table, Likes(a, b) (read 'a' likes 'b'),with Primary Key (a,b). Write a trigger so that when a row is inserted into 'Likes' of the form Likes(X, 'Friendly'), where 'Friendly' is a constant and X can match any value, a tuple that indicates that 'Friendly' Likes X is inserted into Likes (unless it is already in the table). So if Likes('Abe', 'Friendly') is inserted, then Likes('Friendly', 'Abe') is inserted. MOREOVER, for each person, Y, who X Likes, the trigger also inserts Likes ('Friendly', Y) (unless it is already there).

So if Likes('Abe', 'Mary') and Likes('Abe', 'Hua') are in Likes, and Likes('Abe', 'Friendly') is inserted into Likes, then Likes('Friendly', Abe) is inserted, and so is Likes('Friendly', 'Mary') and Likes('Friendly', 'Hua').

Use as close to SQLite syntax as you can.

Implied by SQLite, but ok if added

```
CREATE TRIGGER RinsFriendly
AFTER INSERT ON Likes
REFERENCING NEW ROW AS new
FOR EACH ROW                /* Complete the Trigger definition */
WHEN new.b = 'Friendly'     1 pt
BEGIN

INSERT INTO Likes VALUES ('Friendly', new.a);     1 pt

INSERT INTO Likes(a, b) SELECT 'Friendly', L.b
                        FROM Likes L
                        WHERE new.a = L.a          3 pt

END;
```

Should have BEGIN – END, but no points off if missing

6 minutes

10. (5 pts) Assume that you have a DB with a table, Likes(a, b) (read 'a' likes 'b'),with Primary Key (a,b). Write a trigger so that when a row is inserted into 'Likes' of the form Likes(X, 'Friendly'), where 'Friendly' is a constant and X can match any value, a tuple that indicates that 'Friendly' Likes X is inserted into Likes (unless it is already in the table). So if Likes('Abe', 'Friendly') is inserted, then Likes('Friendly', 'Abe') is inserted. MOREOVER, for each person, Y, who X Likes, the trigger also inserts Likes ('Friendly', Y) (unless it is already there).

So if Likes('Abe', 'Mary') and Likes('Abe', 'Hua') are in Likes, and Likes('Abe', 'Friendly') is inserted into Likes, then Likes('Friendly', Abe) is inserted, and so is Likes('Friendly', 'Mary') and Likes('Friendly', 'Hua').

Use as close to SQLite syntax as you can.

```
CREATE TRIGGER RinsFriendly
AFTER INSERT ON Likes
REFERENCING NEW ROW AS new
FOR EACH ROW                /* Complete the Trigger definition */
WHEN new.b = 'Friendly'
BEGIN
INSERT INTO Likes VALUES (new.b, new.a) WHERE <not redundant>;
INSERT INTO Likes(a, b) SELECT new.b, L.b
                            FROM Likes L
                            WHERE new.a = L.a AND <not redundant>
END;
```

Some answers may generalize the intended answer, by allowing a variable in place of "Friendly" and/or added tests to guard against attempts to insert duplicate rows.

In general, both are fine if done correctly.

11. (3 pts) Consider two tables represented by these schema:

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

Transactions (*AccntNo*: integer, *ProductId*: integer, *date*: string, *amountCharged*: real)

Write a row-level trigger that decrements the appropriate balance in the Accounts table, as defined by matching AccntNo, by the *amountCharged* value of a newly inserted transaction to the Transactions table.

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

```
CREATE TRIGGER UpdateAccountBalance
AFTER INSERT ON Transactions
REFERENCING
   NEW ROW AS New
FOR EACH ROW
BEGIN
   UPDATE Accounts
   SET balance = balance – New.amountCharged
   WHERE Accounts.AccntNo = New.AccntNo;
END;
```

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

Accepting of close (incorrect) syntactic variants,
This includes some SQLite specific aspects

6 minutes

12. (5 pts) Give a UML diagram that captures the following information for a simplified Department of Transportation database.

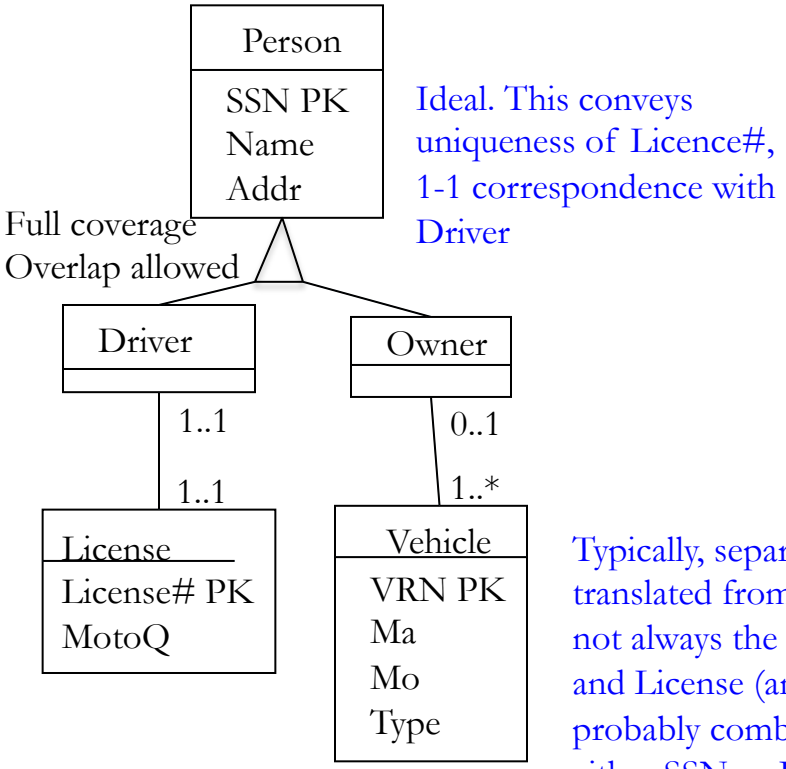
a) A Person can be a licensed vehicle driver and/or a vehicle owner. A person is uniquely identified by SSN, and is also described by name and address. Only persons that are licensed drivers and/or owners are to be recorded in the DB (a driver need not own a vehicle, and an owner need not be a licensed driver).

b) Each vehicle is identified uniquely by vehicle registration number (VRN), with other attributes: make, model, and type (e.g., motorcycle, car, truck).

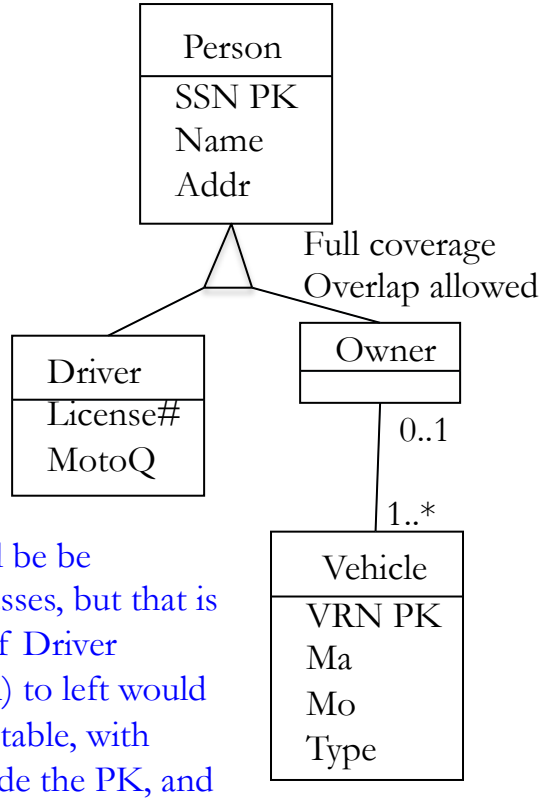
c) Each vehicle is owned by AT MOST one person (i.e., owner).

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).



Ideal. This conveys uniqueness of Licence#, 1-1 correspondence with Driver



This is close, but recognize that it doesn't convey uniqueness of Licence# (which could still be conveyed by UNIQUE in table translation)

Typically, separate SQL tables will be be translated from separate UML classes, but that is not always the case. Translation of Driver and License (and their association) to left would probably combine them into one table, with either SSN or License# being made the PK, and the other UNIQUE.

10 minutes