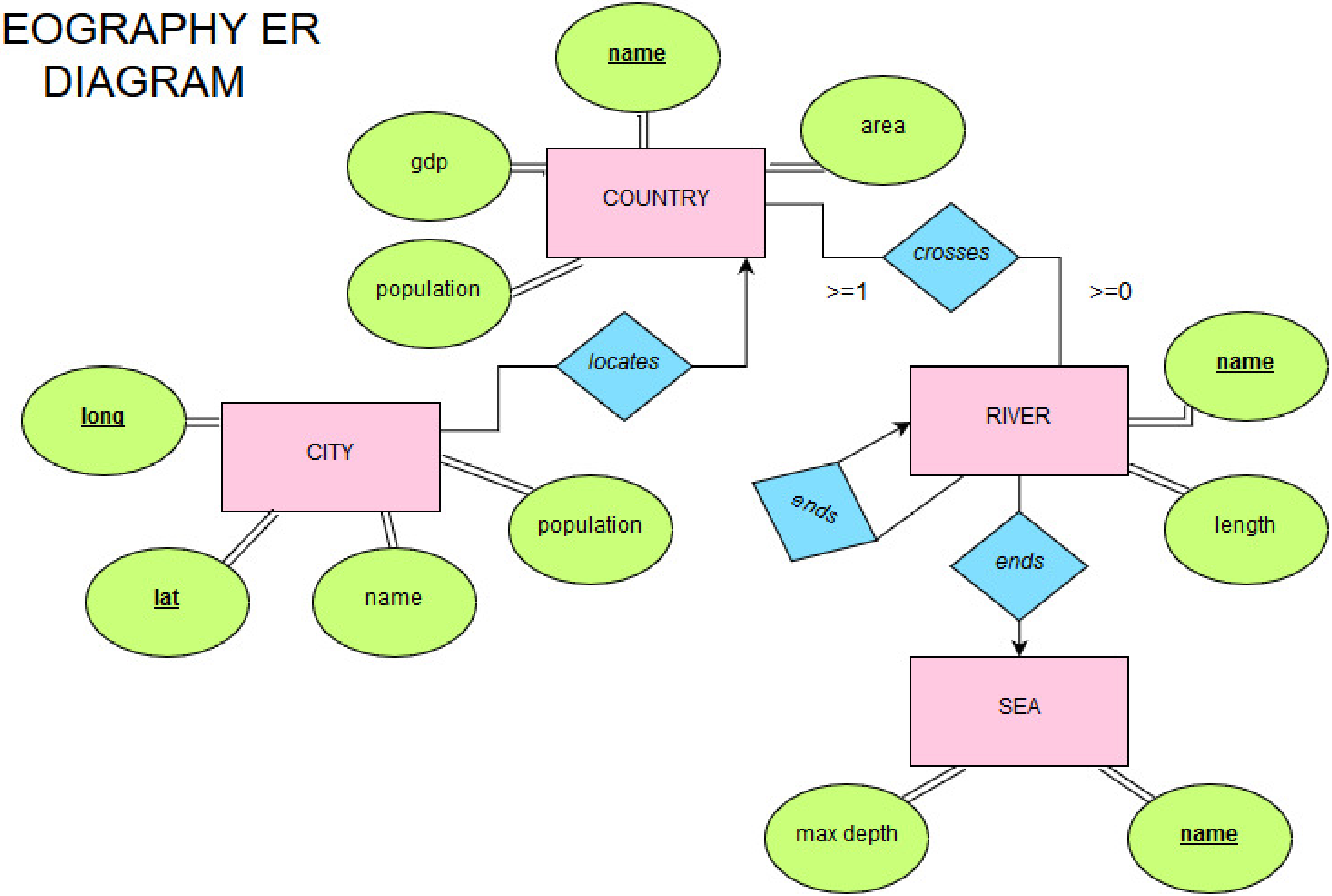
Alex Htut - htutk
1630264

1. GEOGRAPHY ER DIAGRAM



Problem 2

a) Creating tables in SQL

Insurance Company: - Even though *phone* numbers are digits, there is no arithmetic operation to be made. Thus, choose VARCHAR.

```
CREATE TABLE InsuranceCo (
name VARCHAR(20) PRIMARY KEY,
phone VARCHAR(10)
)
```

Vehicle: - Quick Google search suggests that the maximum allowed chars in *licensePlate* is 7.5 in Virginia. - *year* is integer to use for comparisons. - *name* is a foreign key to InsuranceCo Table to represent **Many-to-One** relation between this table (Vehicle) and InsuranceCo. - *ssn* is also a foreign key to Person Table to represent **Many-to-One** relation between this table and Person. Similar to *phone* in InsuranceCo Table, *ssn* is just an ID and thus defined as a VARCHAR.

```
CREATE TABLE Vehicle (
licensePlate VARCHAR(8),
year INTEGER,
name VARCHAR(20) REFERENCES InsuranceCo,
ssn VARCHAR(10) REFERENCES Person
)
```

Person:

```
CREATE TABLE Person (
ssn VARCHAR(10) PRIMARY KEY,
name VARCHAR(20)
)
```

Person.Driver: - *driverSSN* is an inherited primary key (since Person.Driver is a subclass of Person) from Person Table. The syntax `not null REFERENCES Person` ensures that it is the case.

```
CREATE TABLE Driver (
driverSSN VARCHAR(10) not null REFERENCES Person(ssn),
driverID VARCHAR(20),
PRIMARY KEY(driverSSN)
)
```

Person.Driver.NonProessionalDriver: - Again NonProessionalDriver is a subclass of Driver. Thus, its primary key *driverSSN* comes from Driver Table.

```
CREATE TABLE NonProessionalDriver (
driverSSN VARCHAR(10) not null REFERENCES Driver(driverSSN)
)
```

Person.Driver.ProfessionalDriver: - *medicalHistory* type is unknown; set it to VARCHAR.

```
CREATE TABLE ProessionalDriver (
driverSSN VARCHAR(10) not null REFERENCES Driver(driverSSN),
medicalHistory VARCHAR(100)
)
```

Vehicle.Car:

```
CREATE TABLE Car (
licensePlate VARCHAR(8) not null REFERENCES Vehicle,
make VARCHAR(10)
)
```

TypicalCar: - To represent the **Many-to-Many** relation between Car and NonProessionalDriver.

```
CREATE TABLE TypicalCar (
driverSSN VARCHAR(10),
licensePlate VARCHAR(8),
PRIMARY KEY(driverSSN, licensePlate)
)
```

Vehicle.Truck: - Truck, similar to Car, is a subclass of Vehicle. - Assuming the *capacity* is in one unit, define it as an integer for comparisons. - To represent **Many-to-One** relation between Truck and ProessionalDriver, *driverSSN* becomes a foreign key to ProessionalDriver.

```
CREATE TABLE Truck (
licensePlate VARCHAR(8) not null REFERENCES Vehicle,
capacity INTEGER,
driverSSN VARCHAR(10) REFERENCES ProessionalDriver
)
```

b)

Insures has **Many-to-One** relation from Vehicle to InsuranceCo. Thus, we don't need a new table to represent it. Instead, we can insert a foreign key *name* in Vehicle that references InsuranceCo.

c)

Since Drives relation is **Many-to-Many**, we need an additional table that holds the relationship between Car and NonProessionalDriver. On the other hand, since Operates has **Many-to-One** relation from Truck to ProessionalDriver, we can have a foreign key *driverSSN* in Truck that references ProessionalDriver.

Problem 3

a)

```
R(A, B, C, D, E)
FD's: D -> B, CE -> A
```

Closure for FD's:

```
{D}+ = {B, D}
{C, E}+ = {A, C, E}
```

Both closures violate the 'rule'. Pick `D -> B` to decompose further.

We then need to find `Z = {all attritubtes} - {D}+`, we will regard `Z` as `{D}+'`.

```
{D}+' = {A, C, E}
R1({D}+) = R1(B, D)
R2(D U {D}+') = R2(A, C, D, E)
```

Note that `R1(B, D)` is already in BCNF. So, we need to further normalize `R2`.

The closure that violates in `R2` is `{C, E}+ = {A, C, E}`. Calculate `{C, E}+' = {D}`, and decompose `R2` into two more relations:

```
R21({C, E}+) = R21(A, C, E)
R22({C, E} U {C, E}+') = R22(C, D, E)
```

After the FD's is checked once again at this point, both `R21` and `R22` are in BCNF.

Therefore, the final relations are: `R1(B, D)`, `R21(A, C, E)`, and `R22(C, D, E)`

b)

```
S(A, B, C, D, E)
A -> E, BC -> A, DE -> B
```

Closure for FD's:

```
{A}+ = {A, E}
{B, C}+ = {A, B, C, E}
{D, E}+ = {B, D, E}
```

All closures violoate the 'rule'. Choose `{A}+`. Calculate `{A}+' = {B, C, D}`

The decomposed relations are:

```
R1({A}+) = R1(A, E)
R2({A} U {A}+') = R2(A, B, C, D)
```

`R1` is in BCNF.

`{B, C}+ = {A, B, C}` violates. Calculate `{B, C}+' = {D}` Decompose `R2`:

```
R21({B, C}+) = R21(A, B, C)
R22({B, C} U {B, C}+') = R22(B, C, D)
```

Both are in BCNF now.

The final relations are: `R1(A, E)`, `R21(A, B, C)`, and `R22(B, C, D)`

Problem 4

```
R(A, B, C, D)
```

All sets of attributes are closed.

```
No Functional Dependency
```

If any funcitional dependency exists, that set of FD will no longer be closed. For ex. `A -> B` FD is established, the set of attributes `{A}` will have closure `{A}+ = {A, B}`, and no longer be closed.

The only closed sets are {} and {A, B, C, D}

```
A -> B, B -> C, C -> D, D -> A
```

These FD's make sure that every set of attributes (except for the whole set `{A, B, C, D}` and the empty set) will not be closed.

The only closed sets are {}, {A,B}, and {A,B,C,D}

```
A -> B, B -> A
C -> D, D -> C, CD -> A
```

The first pair of FD's guarantees that `{A, B}` is closed. The second trips makes sure that `{C, D}` is not closed.