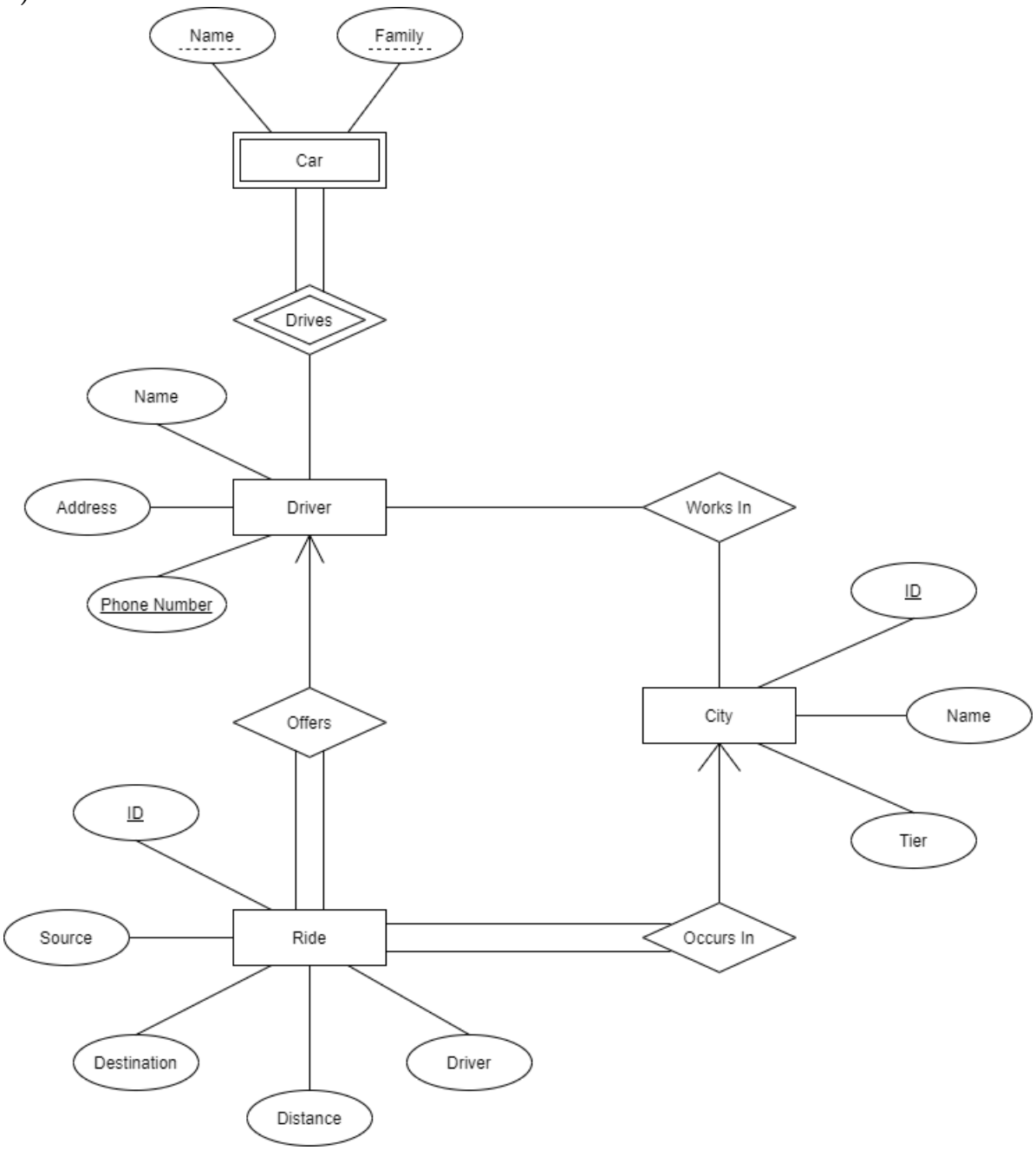
Jason Lai
204995126
Dis 1B

CS 143
February 8, 2021

Homework 3

1 Problem 1

a)



Note: cardinality is expressed using the convention taught in lecture; arrow indicates "one", no arrow indicates "many".

Some assumptions that I made were that every driver has their phone number as their key,

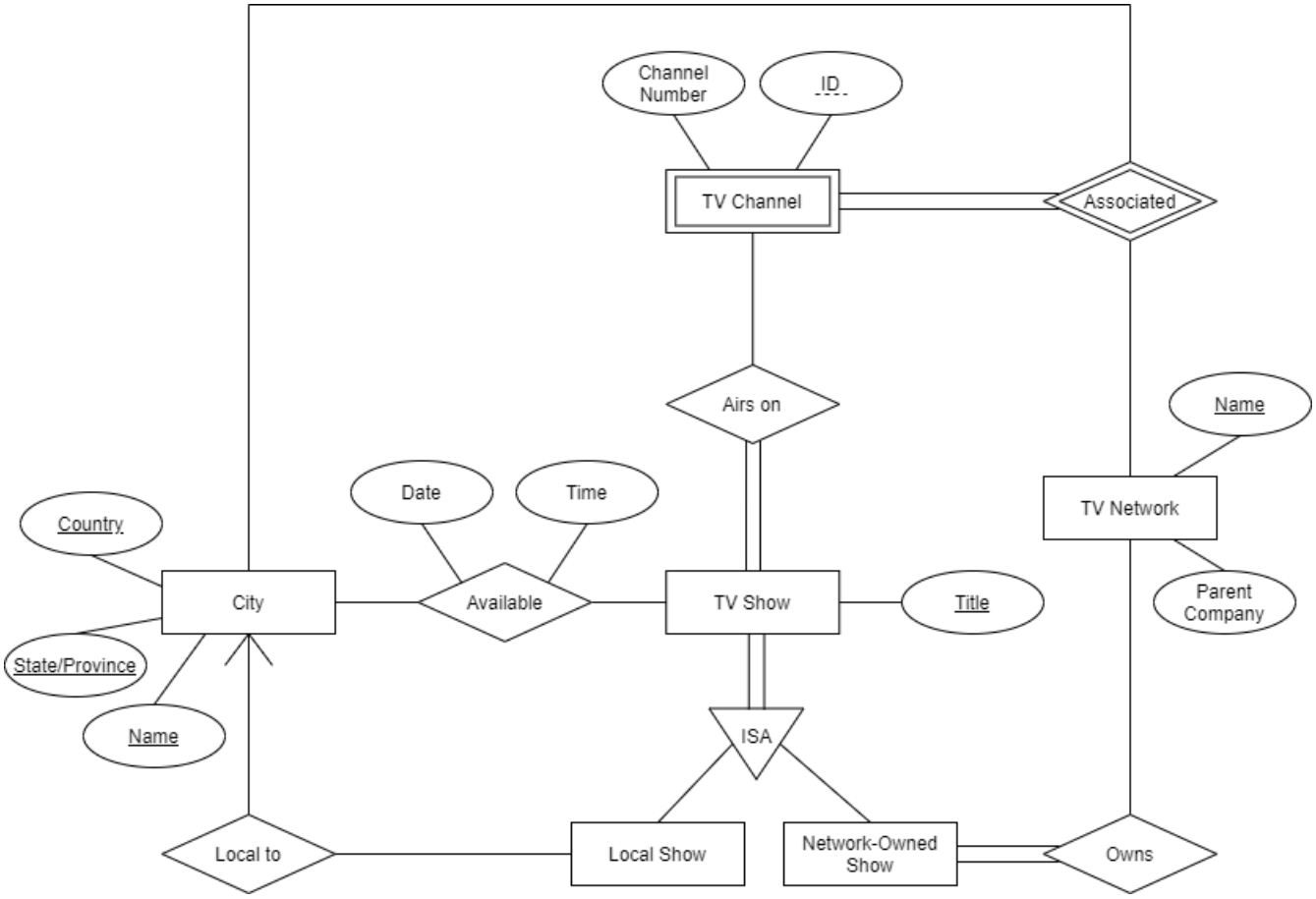
since phone numbers should be unique. I also assumed Nevertaxi can have multiple cars i.e. same family and name, so family and name would not suffice as a key to identify a car, so it is a weak entity set. We use the phone number as part of a specific car's key, and a specific car can have multiple keys, since they can have multiple drivers, however this only works if drivers can't drive multiple of the same type of car. Therefore, if drivers are allowed to drive multiple of the same car e.g. a driver drives multiple distinct BMW sedans, then we cannot express the key constraint of cars, because the attributes of the car are too ambiguous for the many-to-many relationship between drivers and cars. Another assumption I made is that rides need total participation in their relationship with drivers and cities. This is because I assume that a ride must be matched with a driver and city, as rides need someone to execute them and they need a location to occur in, so we see that rides cannot be unmatched without a driver or city and need to total participation. Since cities and rides have IDs, I assumed these to be unique, so they function as keys for those entity sets.

b)

```
CREATE TABLE Driver (
      name VARCHAR(30) NOT NULL,
      address VARCHAR(50),
      phone_number VARCHAR(20) NOT NULL,
      PRIMARY KEY(phone_number)
);
CREATE TABLE Car (
      name VARCHAR(30) NOT NULL,
      family VARCHAR(30) NOT NULL,
      driver VARCHAR(20) NOT NULL,
      PRIMARY KEY(driver, name, family),
      FOREIGN KEY (driver) REFERENCES Driver(phone_number)
);
CREATE TABLE City (
      id INTEGER NOT NULL,
      name VARCHAR(30) NOT NULL,
      tier INTEGER NOT NULL,
      PRIMARY KEY(id)
);
CREATE TABLE Ride (
      id INTEGER NOT NULL,
      source VARCHAR(50) NOT NULL,
      destination VARCHAR(50) NOT NULL,
      distance DECIMAL(5, 2) NOT NULL,
      driver VARCHAR(20) NOT NULL,
      PRIMARY KEY(id)
```

```
);
CREATE TABLE Offers (
      driver VARCHAR(20) NOT NULL,
      rideId INTEGER NOT NULL,
);
CREATE TABLE WorksIn (
      driver VARCHAR(20) NOT NULL,
      cityId INTEGER NOT NULL,
);
CREATE TABLE OccursIn (
      rideId INTEGER NOT NULL,
      cityId INTEGER NOT NULL,
);
```

2 Problem 2



For this problem, I made various assumptions. For one, I assumed that IPs are trademarked or copyrighted, so titles of television shows and names of television networks are unique, therefore they function as key attributes. Furthermore, I assumed that television channels are weak entity sets. They have ID numbers, but only within a certain television network e.g. ABC has various channels listed as ABC1, ABC2, ABC3, etc. Other networks may have channels that also follow this convention, so they are differentiated by the name of the network in addition to the ID number. Furthermore, I assume that local shows are specific to a city and can't be accessed from regions outside that city, hence the one-to-many cardinality between cities and that subclass. I also assume that cities can have access to many television channels and television channels are offered in many cities, and cities are related involved in the association between a channel and a network. As for the specific available of certain shows in certain cities, I have a relationship between cities and shows that determines availability and if available, what the date and time is that a specific show airs.

3 Problem 3

`Programmer(programmer_id, name)`

`TeamLeader(team_id, programmer_id, team_name)`

`WorksWith(programmer_id, team_id, project_id)`

`Project(project_id, project_description)`

Based on the description, examples, and the E/R diagram itself, my reasoning for relations above is as follows:

`Programmer`:
First, programmers have a `programmer_id` attribute that acts as the key for the `Programmer` relation. Next, programmers have a `name` attribute. Despite, the description saying that names uniquely identify programmers, the E/R diagram does not show this attribute as underlined, so I assume that it is not a key; this is also why the `TeamLeader` subclass will not inherit this attribute.

`TeamLeader`:
The `TeamLeader` relation first has a `team_id` attribute, which is the key that uniquely identifies the team leader. Next, we have the `programmer_id` attribute, which is the primary key inherited from the `Programmer` superclass, and identifies which programmers are team leaders. Team leaders are associated with a team name, so we also have the `team_name` attribute.

`WorksWith`:
The `WorksWith` relation has a `programmer_id` attribute which establishes a relationship that a certain programmer has with a specific team and project. The `team_id` attribute describes which team leader, and by extension which team, that the programmer represented by `programmer_id` works with. Finally, the `project_id` attribute describes which project that the team leader `team_id` and programmer `programmer_id` are working on together. Since, programmers can be both a member working on one project and a team leader working on another project i.e. working on different projects simultaneously, we'll have that programmers and team leaders will be associated with multiple `WorksWith` tuples.

`Project`:
An extraneous relation that is implied by the E/R diagram. I assume that `Project` in the E/R diagram refers to an identifier to a project from a project relation, so this is a mock relation to represent that. The `project_id` attribute identifies a specific project and the `project_description` attribute describes that project.