Contents

1 Basic Test Results

```
Extracting Archive:
Archive:  /tmp/bodek.Wpquy9/db/ex1/mohammadgh/rerun/submission
  inflating: ex1.pdf
  inflating: ex1.py
  inflating: README
  inflating: create.sql
  inflating: drop.sql

************************************************************
** Testing that all necessary files were submitted:
README:
    SUBMITTED
create.sql:
    SUBMITTED
drop.sql:
    SUBMITTED
ex1.py:
    SUBMITTED
ex1.pdf:
    SUBMITTED

************************************************************
** Checking for correct README format:

************************************************************
** Testing table creation:
Output:
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE
CREATE TABLE

Number of tables created: 11

************************************************************
** Processing file:
Inserting Movie_person.csv
Output:
COPY 15380

Inserting Producer.csv
Output:
COPY 378

Inserting Actor.csv
Output:
COPY 14365

Inserting Director.csv
Output:
COPY 274

```

```
Inserting Author.csv
Output:
COPY 625

Inserting Oscar.csv
Output:
COPY 93

Inserting Film.csv
Output:
COPY 571

Inserting Winner.csv
Output:
COPY 93

Inserting Nominee.csv
Output:
COPY 478

Inserting Content_rating.csv
Output:
COPY 6

Inserting Genre.csv
Output:
COPY 22


************************************************************
** Testing dropping of tables:
Output:
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE
DROP TABLE

Number of tables dropped: 11
```

2 README

```
mohammadgh,muaz.abdeen
```

3 create.sql

```sql
create table Movie_person
(
    pname varchar(100) primary key
);

create table Actor
(
    aname varchar(100) primary key REFERENCES Movie_person (pname) on delete cascade
);

create table Director
(
    dname varchar(100) primary key REFERENCES Movie_person (pname) on delete cascade
);

create table Author
(
    auname varchar(100) primary key REFERENCES Movie_person (pname) on delete cascade
);

create table Producer
(
    prname varchar(100) primary key REFERENCES Movie_person (pname) on delete cascade
);


create table Oscar
(
    oyear integer primary key CHECK (oyear >= 1900)
);

create table Film
(
    film_id      varchar(100) primary key,
    film_name    varchar(100) not null,
    imdb_rating  float CHECK (0 <= imdb_rating and imdb_rating <= 10),
    imdb_votes   integer      not null,
    duration     integer CHECK (duration > 0),
    release_year integer CHECK (release_year <= osyear + 1),
    osyear       integer,
    FOREIGN KEY (osyear) REFERENCES Oscar (oyear)
);

create table Nominee
(
    film_id varchar(100) primary key REFERENCES Film (film_id) on delete cascade
);

create table Winner
(
    film_id varchar(100) primary key REFERENCES Film (film_id) on delete cascade
);

create table Content_rating
(
    rating varchar(100) primary key
);

create table Genre
```

```
(
    genre_type varchar(100) primary key
);
```

4 drop.sql

```
drop table Actor cascade ;
drop table Director cascade ;
drop table Author cascade ;
drop table Producer cascade ;
drop table Movie_person cascade;
drop table Nominee cascade ;
drop table Winner cascade ;
drop table Film cascade;
drop table Oscar cascade;
drop table Content_rating;
drop table Genre;
```

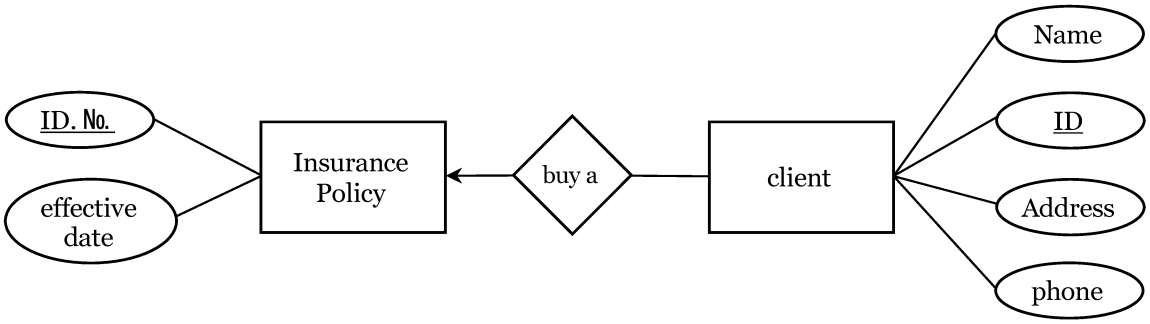
(67506) Databases – Spring 2022 – Exercise (1)

Muaz Abdeen 300575297

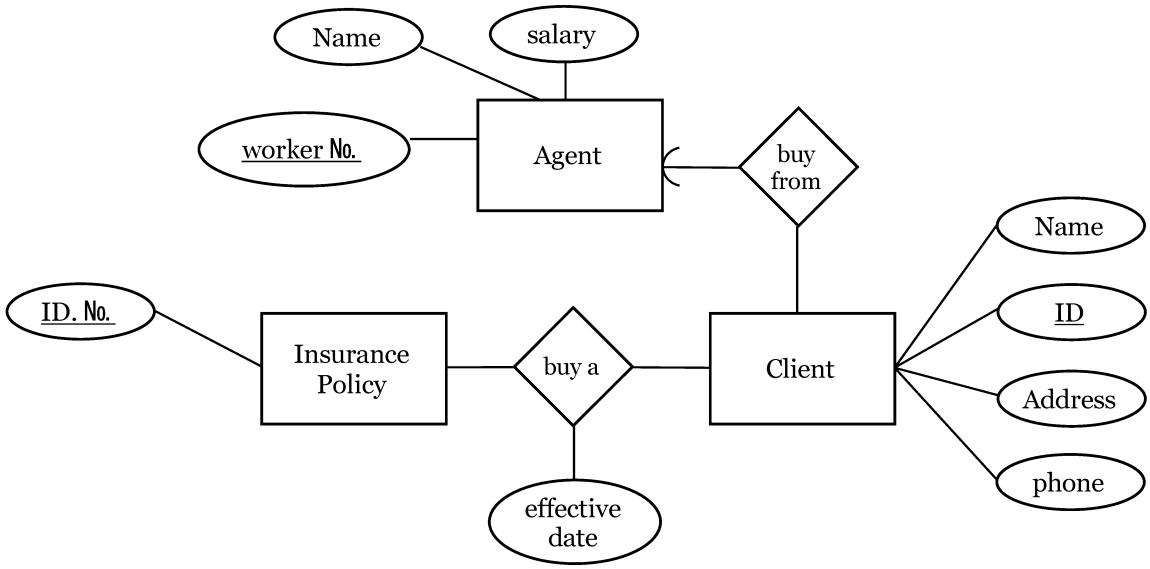
Mohammad Ghanayem 208653220

Question (1):

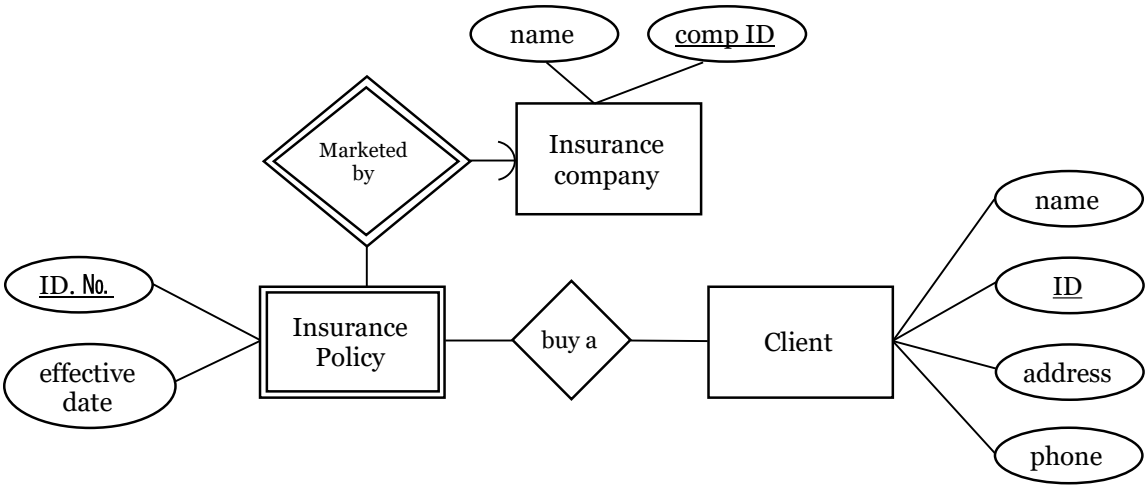
1.



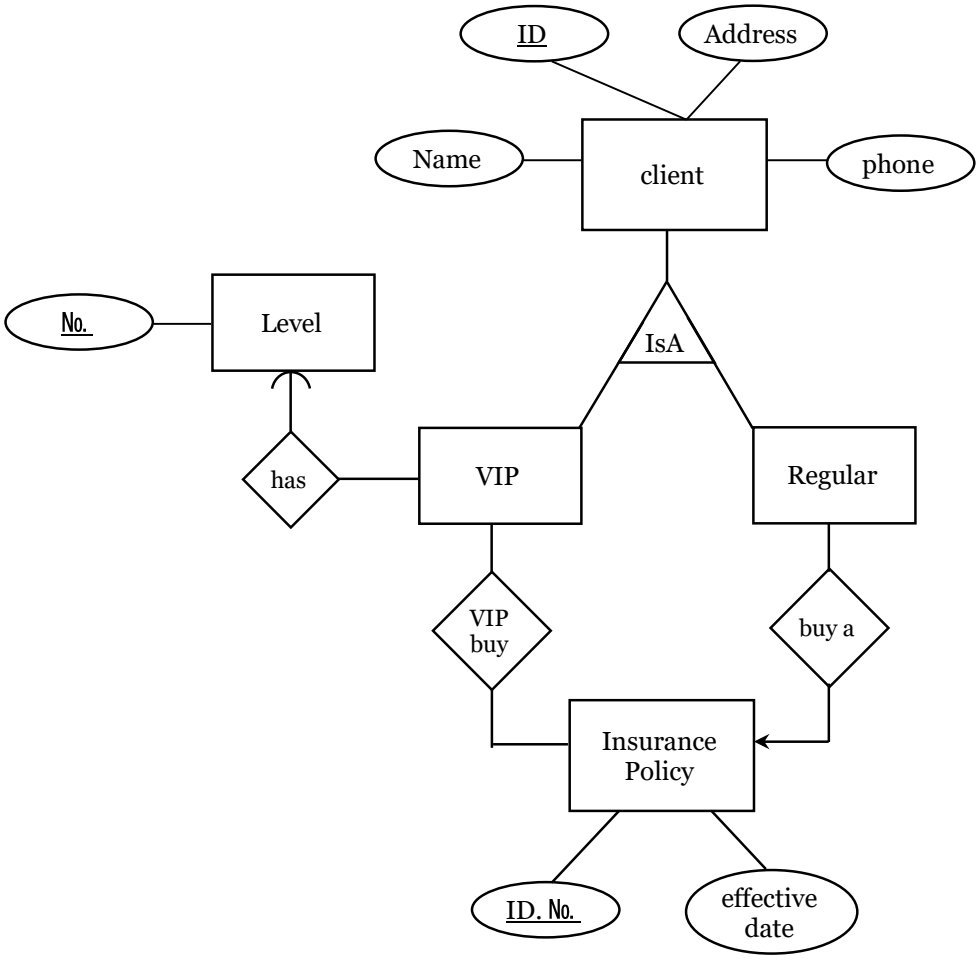
2.



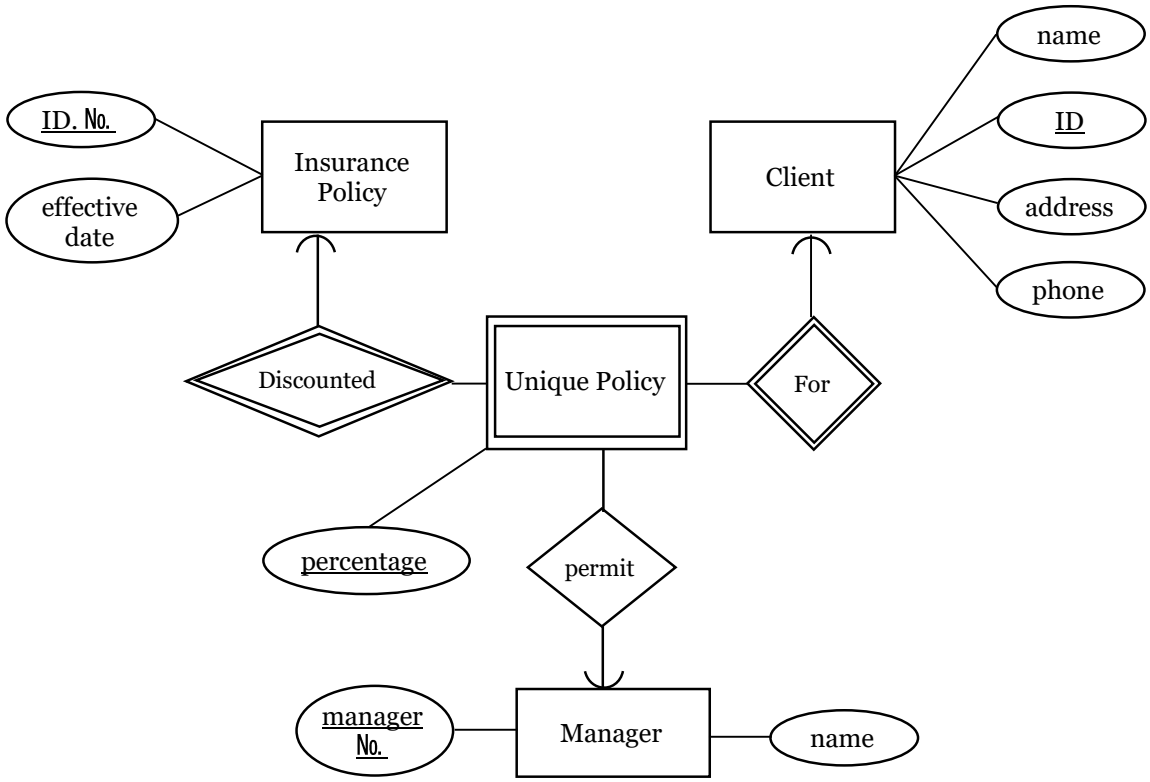
3.


name
comp ID
Marketed by
Insurance company
name
ID. No.
ID
Insurance Policy
buy a
Client
effective date
address
phone

4.


ID
Address
Name
client
phone
No.
Level
IsA
has
VIP
Regular
VIP buy
buy a
Insurance Policy
ID. No.
effective date

5.


ID. No.
effective date
Insurance Policy
Client
name
ID
address
phone
Discounted
Unique Policy
For
percentage
permit
manager No.
Manager
name

Question (2):

a. A(<u>a</u>, b, c)

B(<u>d</u>, e)

R(<u>a</u>, <u>d</u>)

Cannot be determined which set is bigger.

b. A(<u>a</u>, b, c, d)

B(<u>d</u>, e, a)

|A|=|B|

c. A(<u>a</u>, b, c)

B(<u>d</u>, e, a)

C(<u>f</u>)

R(<u>a</u>, <u>d</u>, <u>f</u>)

Cannot be determined which set is bigger.

d. A(<u>a</u>, b)

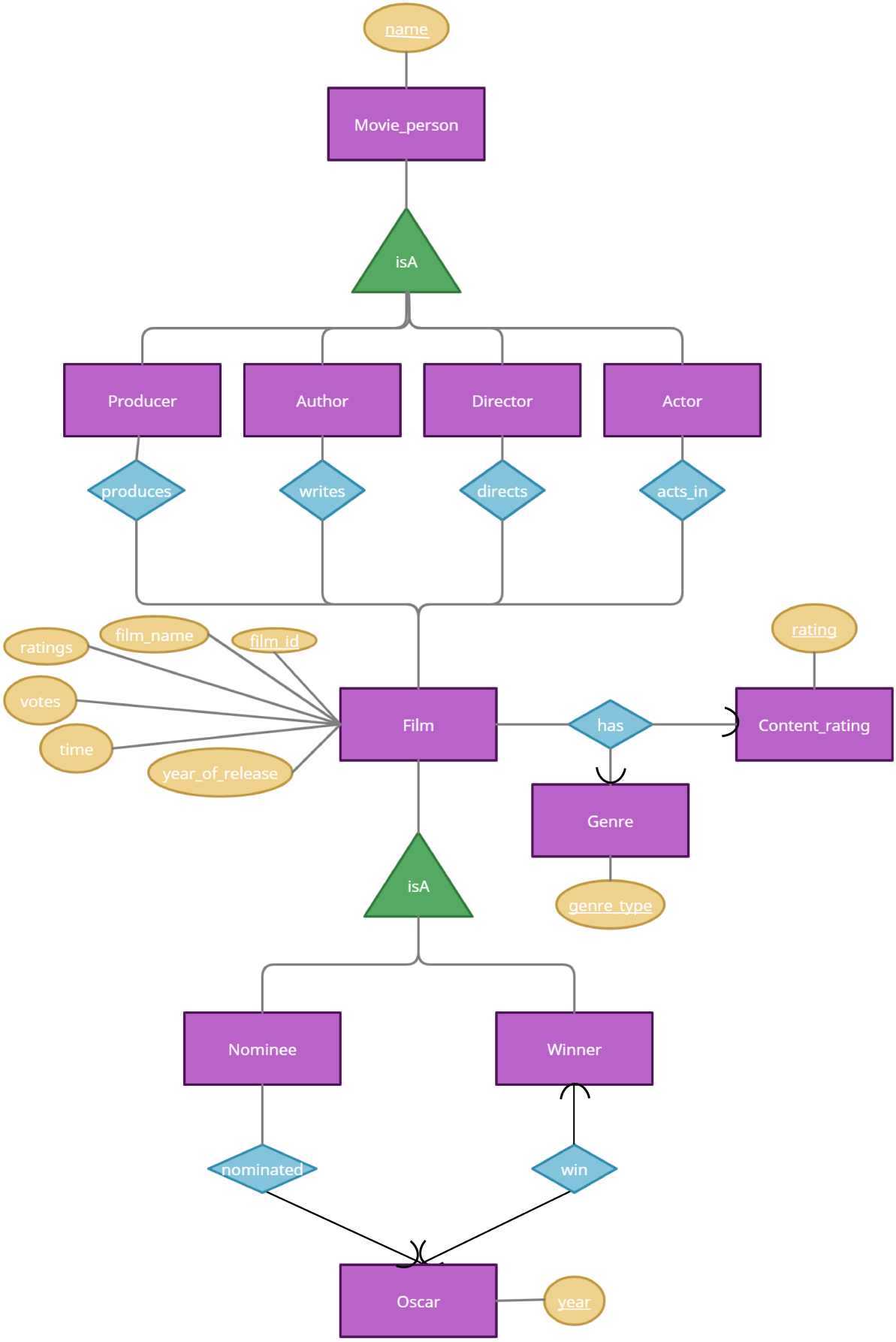
B(<u>a</u>, c)

C(<u>d</u>, <u>a</u>)

|A|>=|B|

Question (3):

a.


name
Movie_person
isA
Producer
Author
Director
Actor
produces
writes
directs
acts_in
ratings
film_name
film_id
votes
time
year_of_release
Film
has
rating
Content_rating
Genre
genre_type
isA
Nominee
Winner
nominated
win
Oscar
year

Explanation for our diagram: -

First, we made a “parent” entity (isA relation) for the entities {Actor, Author, Director, Producer} called “Movie_person”, because all of them has a relation to some movie, the key attribute is the “name” of the movie person and it is unique for each of them, each one of them has a relation between him and the “Film” entity so for each of them we create a relation that differs from other entities, like writes and directs.

We create the entity “Oscar” (for Award (4)) as a normal entity with one attribute that is the key of it named “year”.

To connect the “Oscar” with “Film” we made another two entities with isA relation with “Film” entity, first one is “Winner” that is the Film that wins the Oscar and the other one is “Nominee” that is the nominee for the Oscar, with a relation “wins” and “nominated”, respectively.

Each Oscar in some year has one and just one Winner so we made a circled arrow from Oscar to Winner, and so from Winner to Oscar, so the winner wins just one Oscar in the same year.

Each Nominee nominated for just one Oscar in some year, so we made circled arrow from “Nominee” and “Oscar”.

“Film” entity has its all 6 attributes that written in the description of the exercise, and its key is “film_id” and it is unique for each film, each film has content rating and genre so we made the relation “has” from “Film” entity to “Content_rating” and “Genre” entities, which have “rating” and “genre_type” attributes, respectively, both of these attributes are keys of their entity.

While we are looking at the data we saw that each film has just one genre and one content rating so we made a circled arrow from “Film” entity to “Genre” and “Content_rating” entities

We create an entity called “Film” with 6 attribute and the key is “film_id” and it is unique for each film, the film has content rating and genre, so we made a relation called “has” with two entities called “Content_rating” and “Genre” with “rating” and “genre_type” attributes, respectively, both of them are keys.

b. Movie_person(name)

Actor(name)

Director(name)

Author(name)

Producer(name)

produces(pname, fid)

writes(pname, fid)

directs(pname, fid)

acts_in(pname, fid)

Oscar(year)

Winner(fid)

Nominee(fid)

Film(film_id, film_name, ratings, votes, time, year_of_release)

has(fid, rating, genre_type)

Content_rating(rating)

Genre(genre_type)

c. Not For Submission

d. At Files: create.sql, drop.sql

e. At File: ex1.py

6 ex1.py

```python
import csv
from functools import reduce
from io import TextIOWrapper
from zipfile import ZipFile
import numpy as np

# opens file for oscars table.
# CHANGE!
outfile = open("oscars.csv", 'w', )
outwriter = csv.writer(outfile, delimiter=",", quoting=csv.QUOTE_NONE)


# return the list of all tables
def get_names() -> list:
    return ["Movie_person", "Producer", "Actor", "Director", "Author",
            "Oscar", "Film", "Winner", "Nominee", "Content_rating", "Genre"]


out_files = [open(f"{name}.csv", 'w', ) for name in get_names()]
out_writers = [csv.writer(out_file, delimiter=",", quoting=csv.QUOTE_NONE)
               for out_file in out_files]
attr_indices = [[3, 11, 12, 13], [3], [13], [11], [12], [2],
                [14, 1, 8, 9, 6, 5, 2], [14], [14], [10], [7]]
seen = [set() for i in range(len(get_names()))]


# def setup():
attributes = [["pname"], ["prname"], ["aname"], ["dname"], ["auname"], ["oyear"],
              ["film_id", "film_name", "imdb_rating", "imdb_votes", "duration",
               "release_year", "osyear"], ["film_id"], ["film_id"],
              ["rating"], ["genre_type"]]
for idx in range(len(out_writers)):
    out_writers[idx].writerow(attributes[idx])


# def cleanup():
#     for file in out_files:
#         file.close()


def write_attributes(table, row):
    # process Oscar, Film, Winner, Nominee tables (have neither && no NULL)
    if table in [5, 6, 7, 8]:
        attr_values = row[attr_indices[table]]
        if tuple(attr_values) in seen[table]:
            return
        seen[table].add(tuple(attr_values))
        if row[4] == 'Nominee' and table == 8:
            out_writers[table].writerow(attr_values)
        elif row[4] == 'Winner' and table == 7:
            out_writers[table].writerow(attr_values)
        elif table != 8 and table != 7:
            out_writers[table].writerow(attr_values)
    # process other tables which may have && or NULL
    else:
        attr_values = row[attr_indices[table]]
        attr_values[attr_values == ''] = 'NULL'
        attr_values = np.char.split(attr_values, sep='&&')
        attr_values = np.array([np.array(lst) for lst in attr_values])
```

```
        attr_values = reduce(np.union1d, attr_values)

        for value in attr_values:
            value = value.strip()
            if value in seen[table]:
                continue
            seen[table].add(value)
            out_writers[table].writerow([value])


INITIAL_ROW = ['', 'Film', 'Oscar Year', 'Film Studio/Producer(s)', 'Award', 'Year of Release',
               'Movie Time', 'Movie Genre', 'IMDB Rating', 'IMDB Votes', 'Content Rating',
               'Directors', 'Authors', 'Actors', 'Film ID']


# process_row should splits row into the different csv table files
# CHANGE!!!
def process_row(row):
    row = np.array(row)
    if np.all(row == INITIAL_ROW):
        outwriter.writerow(row)
        return

    for table in range(len(get_names())):
        write_attributes(table, row)

    outwriter.writerow(row)


# process_file goes over all rows in original csv file, and sends each row to process_row()
# DO NOT CHANGE!!!
def process_file():
    with ZipFile('archive.zip') as zf:
        with zf.open('oscars_df.csv', 'r') as infile:
            reader = csv.reader(TextIOWrapper(infile, 'utf-8'))
            for row in reader:
                # remove some of the columns
                chosen_indices = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 13, 14, 15, 16, 29]
                row = [row[index] for index in chosen_indices]

                # change "," into && in list values
                lists_values_indices = [7, 11, 12, 13]
                for list_value_index in lists_values_indices:
                    row[list_value_index] = row[list_value_index].replace(',', '&&')

                # pre-process : remove all quotation marks from input and turns NA into null
                # value ''.
                row = [v.replace(',', '') for v in row]
                row = [v.replace("'", '') for v in row]
                row = [v.replace('"', '') for v in row]
                row = [v if v != 'NA' else "" for v in row]

                # In the first years of oscars in the database they used "/" for example 1927/28,
                # so we will change these.
                row[2] = row[2].split("/")[0]

                # In 1962 two movies were written as winners, then we change one of them to nominee.
                if row[4] == "Winner" and row[2] == "1962" and row[
                    14] == "8d5317bd-df12-4f24-b34d-e5047ef4665e":
                    row[4] = "Nominee"

                # In 2020 Nomadland won and marked as nominee by mistake.
                if row[2] == "2020" and row[1] == "Nomadland":
                    row[4] = "Winner"

                process_row(row)

    # flush and close the file. close all of your files.
```

```python
    outfile.close()
    for file in out_files:
        file.close()



# return a list of all the inner values in the given list_value.
# you should use this to handle value in the original table which
# contains an inner list of values.
# DO NOT CHANGE!!!
def split_list_value(list_value):
    return list_value.split("&&")


if __name__ == "__main__":
    process_file()
```