Nisharg Gosai
IDMP Assignment 3

OLAP

Write the following queries in the university database:

1. Based on the number of total students enrolled in each course, rank the courses from most popular to least popular.
The result should include the columns: course_id, course_title, students_num, and popularity_rank.

```
WITH cte AS (
   SELECT c.course_id,c.title as course_title,COUNT(s.ID) as students_num
   FROM (course c JOIN takes t ON c.course_id = t.course_id) JOIN student s ON t.ID =
s.ID
   GROUP BY c.course_id,c.title
)
SELECT course_id, course_title, students_num,
   DENSE_RANK() OVER (ORDER BY students_num DESC) as popularity_rank
FROM cte;
```

Result Grid Filter Rows: Search Export:

course_id	course_title	students_num	popularity_rank
CS-101	Intro. to Computer Science	7	1
CS-190	Game Design	2	2
CS-315	Robotics	2	2
CS-319	Image Processing	2	2
CS-347	Database System Concepts	2	2
BIO-101	Intro. to Biology	1	3
BIO-301	Genetics	1	3
EE-181	Intro. to Digital Systems	1	3
FIN-201	Investment Banking	1	3
HIS-351	World History	1	3
MU-199	Music Video Production	1	3
PHY-101	Physical Principles	1	3

2. Analyze the academic performance of students across courses and departments.
Show the average grade by department, courses within that department, and overall average grade across all departments and courses.
Use the following conversions from grade letters to numbers:
['A','A+','A-'] → 100,
['B','B+','B-'] → 90,
['C','C+','C-'] → 80,
['D','D+','D-'] → 60, and 0 for all the other letters.
The result should include the columns: department, course, grades_average.

```
WITH cte AS(
        SELECT dept_name,course.course_id,
   CASE
                WHEN grade IN('A','A+','A-') THEN 100
                WHEN grade IN('B','B+','B-') THEN 90
                WHEN grade IN('C','C+','C-') THEN 80
                WHEN grade IN('D','D+','D-') THEN 60
                ELSE 0 END AS grade_new
   FROM takes INNER JOIN course ON takes.course_id=course.course_id)

SELECT dept_name AS department,course_id AS course, AVG(grade_new) AS grades_average
FROM cte
GROUP BY dept_name,course_id WITH ROLLUP;
```

departme...	course	grades_average
Biology	BIO-101	100.0000
Biology	BIO-301	0.0000
Biology	NULL	50.0000
Comp. Sci.	CS-101	78.5714
Comp. Sci.	CS-190	95.0000
Comp. Sci.	CS-315	95.0000
Comp. Sci.	CS-319	95.0000
Comp. Sci.	CS-347	100.0000
Comp. Sci.	NULL	88.0000
Elec. Eng.	EE-181	80.0000
Elec. Eng.	NULL	80.0000
Finance	FIN-201	80.0000
Finance	NULL	80.0000
History	HIS-351	90.0000
History	NULL	90.0000
Music	MU-199	100.0000
Music	NULL	100.0000
Physics	PHY-101	90.0000
Physics	NULL	90.0000
NULL	NULL	84.5455

3. For each semester identify the top 3 students that received the highest grades in that semester (across all the courses they took in that semester).
The result should include the columns: year, semester, student_name, grade.

Hint: First write a Common Table Expression (CTE) that uses the ROW_NUMBER() window function to partition the grades by year and semester and assign a number to each student based on their performance in that semester. Then, in the main query use the row numbers assigned to the students inside a WHERE clause in order to select the top 3 students in each partition.

```
WITH cte AS(
        SELECT * ,
   ROW_NUMBER() OVER(PARTITION BY semester,year ORDER BY grade) AS rowno
   FROM takes)
SELECT year,semester,name,grade
FROM cte INNER JOIN student ON student.id =cte.ID
WHERE rowno<=3;
```

year	semester	name	grade
2017	Fall	Zhang	A
2017	Fall	Shankar	A
2017	Fall	Brown	A
2017	Spring	Shankar	A
2017	Spring	Williams	B+
2017	Spring	Aoi	C
2018	Spring	Shankar	A
2018	Spring	Brown	A
2018	Spring	Sanchez	A-
2017	Summer	Tanaka	A
2018	Summer	Tanaka	NULL

Web Scraping

Write a Python program to download IMDB's top 250 movies from https://www.imdb.com/chart/top and load them into a data frame. For each movie show its title, director name, list of actors, release year, and IMDB rating.

```
import requests
import pandas as pd
from bs4 import BeautifulSoup

response =
requests.get("https://www.imdb.com/search/title/?groups=top_100&sort=user_rating,des
c")
soup = BeautifulSoup(response.text, 'html.parser')

titles = []
years = []
ratings = []
directors = []
actors_list = []

for movie in soup.find_all('div', class_='lister-item-content')[:50]:
    header = movie.find('h3', class_='lister-item-header')

    titles.append(header.find('a').get_text())

    years.append(header.find('span', class_='lister-item-year').get_text())

    ratings.append(movie.find('div', class_='ratings-imdb-rating').find('strong').get_text())

    director_stars = movie.find_all('p')[2].get_text(strip=True)
    director_stars = director_stars.replace('|', '').split('Stars:')
    directors.append(director_stars[0].replace('Director:', '').replace('Directors:', '').strip())

    actors = director_stars[1].split(',') if len(director_stars) > 1 else 'N/A'
    actors_list.append(', '.join(actors))

df = pd.DataFrame({
    'Title': titles,
    'Release Year': years,
    'IMDb Rating': ratings,
    'Director': directors,
    'Actors': actors_list
})
df
```

```
df
```

Out[13]:

	Title	Director	Actors	Release Year	IMDb Rating
0	The Shawshank Redemption	Frank Darabont	Tim Robbins, Morgan Freeman, Bob Gunton, Willi...	(1994)	9.3
1	The Godfather	Francis Ford Coppola	Marlon Brando, Al Pacino, James Caan, Diane Ke...	(1972)	9.2
2	The Dark Knight	Christopher Nolan	Christian Bale, Heath Ledger, Aaron Eckhart, M...	(2008)	9.0
3	Schindler's List	Steven Spielberg	Liam Neeson, Ralph Fiennes, Ben Kingsley, Caro...	(1993)	9.0
4	The Lord of the Rings: The Return of the King	Peter Jackson	Elijah Wood, Viggo Mortensen, Ian McKellen, Or...	(2003)	9.0
5	12 Angry Men	Sidney Lumet	Henry Fonda, Lee J. Cobb, Martin Balsam, John ...	(1957)	9.0
6	The Godfather Part II	Francis Ford Coppola	Al Pacino, Robert De Niro, Robert Duvall, Dian...	(1974)	9.0
42	Léon: The Professional	Luc Besson	Jean Reno, Gary Oldman, Natalie Portman, Danny...	(1994)	8.5
43	The Lion King	Roger Allers,Rob Minkoff	Matthew Broderick, Jeremy Irons, James Earl Jo...	(1994)	8.5
44	The Usual Suspects	Bryan Singer	Kevin Spacey, Gabriel Byrne, Chazz Palminteri,...	(1995)	8.5
45	The Intouchables	Olivier Nakache,Éric Toledano	François Cluzet, Omar Sy, Anne Le Ny, Audrey F...	(2011)	8.5
46	American History X	Tony Kaye	Edward Norton, Edward Furlong, Beverly D'Angel...	(1998)	8.5
47	The Pianist	Roman Polanski	Adrien Brody, Thomas Kretschmann, Frank Finlay...	(2002)	8.5
48	Casablanca	Michael Curtiz	Humphrey Bogart, Ingrid Bergman, Paul Henreid,...	(1942)	8.5
49	Once Upon a Time in the West	Sergio Leone	Henry Fonda, Charles Bronson, Claudia Cardinal...	(1968)	8.5

XML and XPath

Download the file recipes.xml from Canvas. The file contains a collection of recipes.

• Recipes consist of ingredients, steps for preparation, possibly some comments, and a specification of its nutrition.

• An ingredient can be simple or composite.

• A simple ingredient has a name, an amount (possibly unspecified), and a unit (unless the amount is dimensionless).

• A composite ingredient is recursively a recipe with ingredients and preparation.

1. Write a Python program to read the XML file and display a table with the title of each recipe, the names of its ingredients, and the number of calories.

```
import xml.etree.ElementTree as ET
import pandas as pd

def extract_ingredients(recipe):
    ingredients = []
    for ingredient in recipe.findall('ingredient'):

        name = ingredient.get('name')
        if name is None:

            name_element = ingredient.find('name')
            if name_element is not None:
                name = name_element.text
        if name is not None:
            ingredients.append(name)
    return ingredients


tree = ET.parse('/Users/nisharggosai/Desktop/idmp ass3/recipes.xml')
root = tree.getroot()


recipes = []


for recipe in root.findall('recipe'):
    title = recipe.findtext('title') or "N/A"


    ingredients = extract_ingredients(recipe)
    ingredients_str = "||".join(ingredients)
```

```
    nutrition_element = recipe.find('.//nutrition')
    calories = nutrition_element.get('calories') if nutrition_element is not None else "N/A"


    recipes.append([title, ingredients_str, calories])


df = pd.DataFrame(recipes, columns=['Title', 'Ingredients', 'Calories'])

df
```

Out[25]:

	Title	Ingredients	Calories
0	Beef Parmesan with Garlic Angel Hair Pasta	beef cube steak||onion, sliced into thin rings...	1167
1	Ricotta Pie	filling||dough||milk	349
2	Linguine Pescadoro	linguini pasta||sauce	532
3	Zuppa Inglese	egg yolks||milk||Savoiardi biscuits||sugar||Al...	612
4	Cailles en Sarcophages	pastry||filling||package phyllo dough||egg whi...	8892

2. Write the following XPath queries:

(a) Find the titles of all recipes.

```
tree.xpath('//recipe/title/text()')
```

```
['Beef Parmesan with Garlic Angel Hair Pasta',
 'Ricotta Pie',
 'Linguine Pescadoro',
 'Zuppa Inglese',
 'Cailles en Sarcophages']
```

(b) Find the titles of recipes that use olive oil.(assuming olive oil is not ingredient of ingredient/not nested)

```
tree.xpath("//recipe[ingredient/@name='olive oil']/title/text()")
```

```
['Beef Parmesan with Garlic Angel Hair Pasta']
```

(c) Find the titles of all recipes with less than 500 calories.

```
tree.xpath("//recipe[nutrition/@calories<500]/title/text()")
```

```
['Ricotta Pie']
```

(d) Find the amount of sugar needed for Zuppa Inglese.

```
tree.xpath("//recipe[title='Zuppa Inglese']/ingredient[@name='sugar']/@amount")
```

```
['0.75']
```

(e) Find the titles of all recipes that require 4 steps.

```
for recipe in tree.xpath("//*/recipe"):
    if len(recipe.xpath(".//step"))==4:
        print(recipe.xpath("./title/text()"))
```

```
['Beef Parmesan with Garlic Angel Hair Pasta']
```

(f) Find the names of all ingredients that are used to make other ingredients.

```
In [88]: tree.xpath("//ingredient/@name")

Out[88]: ['beef cube steak',
          'onion, sliced into thin rings',
          'green bell pepper, sliced in rings',
          'Italian seasoned bread crumbs',
          'grated Parmesan cheese',
          'olive oil',
          'spaghetti sauce',
          'shredded mozzarella cheese',
          'angel hair pasta',
          'minced garlic',
          'butter',
          'filling',
          'ricotta cheese',
          'eggs',
          'white sugar',
          'vanilla extract',
          'semisweet chocolate chips',
          'dough',
          'flour',
          'baking powder',
          'white sugar',
          'shortening',
          'eggs, lightly beaten',
          'vanilla extract',
          'milk',
          'linguini pasta',
          'sauce',
          'olive oil',
          'minced cloves of garlic',
          'Italian seasoning',
          'dried thyme',
          'crushed red pepper flakes',
          'crushed tomatoes',
          'black olives, drained',
          'whole baby clams',
          'minced clams, with juice',
          'small salad shrimp',
          'scallops',
          'lemon zest',
          'salt',
          'ground black pepper',
          'egg yolks',
          'milk',
          'Savoiardi biscuits',
          'sugar',
          'Alchermes liquor',
          'lemon zest',
          'flour',
          'fresh whipping cream',
          'pastry',
          'chilled unsalted butter',
          'flour',
          'salt',
          'ice water',
          'filling',
          'baked chicken',
          'marinated chicken',
          'small chickens, cut up',
          'Herbes de Provence',
          'dry white wine',
          'orange juice',
          'minced garlic',
          'truffle oil',
          'stock',
          'chicken wings, giblets, and kidney',
          'onions, peeled',
          'carrots, peeled and cut lengthwise',
          'celery, cut lengthwise',
          'bay leaf',
          'small bunch parsley',
          'whole peppercorns',
          'salt',
          'sauteed mushrooms',
          'white button mushrooms',
          'butter',
          'dry white wine',
          'minced garlic',
          'minced shallots',
          'sauce',
          'chicken juices',
          'mushroom juices',
          'sherry',
          'flour',
          'butter',
          'package phyllo dough',
          'egg whites, lightly beaten']
```

(g) Find the names of all ingredients for which you need other ingredients.

```
ingredient_names_xpath = tree.xpath("//ingredient[ingredient]/@name")
unique_ingredient_names = list(set(ingredient_names_xpath))
for ingredient in unique_ingredient_names:
    print(ingredient)
```

```
marinated chicken
filling
dough
baked chicken
sauteed mushrooms
stock
sauce
pastry
```

(h) Find the names of the first three ingredients in each recipe.

```
[recipe.xpath("ingredient[position() <= 3]/@name") for recipe in tree.xpath('//recipe')]
```

```
[['beef cube steak',
  'onion, sliced into thin rings',
  'green bell pepper, sliced in rings'],
 ['filling', 'dough', 'milk'],
 ['linguini pasta', 'sauce'],
 ['egg yolks', 'milk', 'Savoiardi biscuits'],
 ['pastry', 'filling', 'package phyllo dough']]
```

JSON and Web API

The Open Directions API provides a free web service that returns JSON-formatted driving directions between locations. Description of the service and its parameters can be found at https://developer.mapquest.com/documentation/directions-api.

In order to use the service, you will first need to obtain a free API key from https://developer.mapquest.com.

Write a Python program that gets from the user an origin and a destination location, and displays the driving instructions to get from the origin to the destination. For each leg of the route, show the driving instruction, distance and time.

```
import requests
import pandas as pd
import json

api_key = "Q8bp0NL2HD9qAX82ByKfgjnwLzmjbgke"  # API key
base_url = "http://www.mapquestapi.com/directions/v2/route"

origin = input("Enter the origin location: ")
destination = input("Enter the destination location: ")

params = {
   "key": api_key,
   "from": origin,
   "to": destination
   }

response = requests.get(base_url, params=params)

if response.status_code == 200:
   directions = response.json()
   route = directions['route']
   legs = route['legs']

   data = {
      "Instruction": [],
      "Distance (miles)": [],
      "Time (minutes)": []
   }
```

```
    for i, leg in enumerate(legs, 1):
        for maneuver in leg['maneuvers']:
            instruction = maneuver['narrative']
            distance = maneuver['distance']
            time = maneuver['time'] / 60
            data['Instruction'].append(instruction)
            data['Distance (miles)'].append(distance)
            data['Time (minutes)'].append(time)

    df = pd.DataFrame(data)
    print(df)
else:
    print(f"Error: Unable to fetch directions. Status code: {response.status_code}")
```

```
Enter the origin location: 75 saint alphonsus street
Enter the destination location: target boylston street
                                       Instruction  Distance (miles)  \
0        Head toward St Alphonsus St. Go for 125 ft.            0.0236
1    Turn right onto St Alphonsus St. Go for 0.1 mi.            0.1143
2         Turn right onto Tremont St. Go for 0.2 mi.            0.2019
3  Turn left onto Huntington Ave (RT-9). Go for 4...            4.6820
4  Keep left onto Boylston St (RT-9 W). Go for 1....            1.5174
5        Keep right onto Boylston St. Go for 0.2 mi.            0.1920
6         Turn left onto Chestnut St. Go for 148 ft.            0.0280
7        Turn right onto Boylston St. Go for 167 ft.            0.0317
8  Arrive at Boylston St. Your destination is on ...            0.0000

   Time (minutes)
0        0.266667
1        0.416667
2        1.600000
3       11.266667
4        2.266667
5        0.516667
6        0.100000
7        0.116667
8        0.000000
```