DS 5110: Introduction to Data Management and Processing

Sample Midterm Exam (Solutions)

- You have 3 hours.
- The exam is closed book.
- You are allowed two letter-sized pages of notes (both sides).
- No electronic devices are allowed.
- Please read all questions carefully before answering them.
- Good luck!

Name: _____

NEU ID (optional): _____

1 Multiple-Choice Questions (32 points)

Circle **ALL the correct choices**: there may be more than one correct choice, but there is always at least one correct choice. **NO partial credit**: the set of all the correct answers must be checked. There are 8 multiple choice questions worth 4 points each.

1. What will be the output of the following code?

```
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6]])
print(arr > 3)
```

   (a) `[False, True]`

   (b) `[False, False, False, True, True, True]`

   (c) `[[False, False, False], [True, True, True]`

   (d) `[4, 5, 6]`

   (e) There will be an error.

   (b). The expression `arr > 3` creates a boolean array, where each element of the original array is compared to 3.

2. What will be the output of the following code?

```
import numpy as np

arr = np.array([[1, 2], [3, 4], [5, 6]])
result = arr * np.array([10, 20]).reshape(-1, 1)
print(result)
```

   (a) `[[10, 40], [30, 80], [50, 120]]`

   (b) `[[10, 30, 50], [40, 80, 120]]`

   (c) `[[10, 20], [30, 40], [50, 60]]`

   (d) There will be an error.

   (d). Broadcasting in NumPy works only when the dimensions are compatible. In this case, the shape of `arr` is (3, 2) and the reshaped array is (2, 1), which cannot be broadcasted together, leading to a broadcasting error.

3. Which of the following statements about Matplotlib components is correct?

   (a) The figure is the entire area that holds all elements of the plot.

   (b) An axes object can show multiple graphs within the same plotting area.

   (c) The legend can only be placed in the upper right corner of the plot.

   (d) The function `plt.subplots(3, 2)` creates six axes objects.

   (e) The `label()` function is used to set the title of the plot.

(a), (b) and (d).

4. Consider the following DataFrame:

```
import pandas as pd

data = {'A': [1, 2, None, 4],
        'B': [5, None, 7, 8],
        'C': [None, None, 10, None]}
df = pd.DataFrame(data)
```

You want to fill the `None` values in the DataFrame using the mean of each row. Which of the following options will give the correct output?

(a) `df.fillna(df.mean(axis=1))`

(b) `df.apply(lambda x: x.fillna(x.mean()), axis=0)`

(c) `df.apply(lambda x: x.fillna(x.mean()), axis=1)`

(d) `df.apply(lambda x: x.fillna(df.mean(axis=1)), axis=0)`

(c). This option works because it applies a row-wise operation (`axis=1`) using `apply()`, replacing `None` values with the mean of each row through `fillna(x.mean())`.

5. Which of the following statements about foreign keys is true?

(a) Foreign keys establish referential integrity constraints by preventing the entry of values in the child table that do not exist in the parent table.

(b) A foreign key can only have unique values.

(c) A foreign key cannot be part of a primary key.

(d) A foreign key can have `NULL` values.

(e) The `ON DELETE CASCADE` option ensures that when a record in the parent table is deleted, the corresponding records in the child table are also deleted.

(a), (d), (e)

6. Suppose that $S(A, \ldots)$ is a table containing m rows and $T(B, \ldots)$ is a table containing n rows. Assume that A is the primary key of S and B is a foreign key reference to $S(A)$. Suppose that we perform an inner join between S and T on $S.A = T.B$. The maximum number of rows that can be in the output is:

(a) m

(b) n

(c) $m + n$

(d) mn

(e) $\max(m, n)$

(b). When we perform an inner join $S.A = T.B$, each row in T can match at most one row of S (since the values of the primary key A are unique), therefore the maximum number of rows we could have in the result is n.

7. Assume that we have executed the following SQL statements:

```
CREATE TABLE s(a INT, b INT);
CREATE TABLE r(a INT, b INT, c INT);
INSERT INTO s(a,b) VALUES (1,2);
INSERT INTO s(a,b) VALUES (3,3);
INSERT INTO s(a,b) VALUES (1,2);
INSERT INTO s(a,b) VALUES (1,4);
INSERT INTO s(a,b) VALUES (3,4);
INSERT INTO r(a,b,c) VALUES (1,2,10);
INSERT INTO r(a,b,c) VALUES (3,3,3);
INSERT INTO r(a,b,c) VALUES (1,7,3);
INSERT INTO r(a,b,c) VALUES (1,2,8);
INSERT INTO r(a,b,c) VALUES (3,3,5);
INSERT INTO r(a,b,c) VALUES (1,1,10);
INSERT INTO r(a,b,c) VALUES (2,2,10);
```

What will be the result of the following query:

```
SELECT c
FROM r
GROUP BY c
HAVING COUNT(c) = (SELECT(COUNT(DISTINCT a))
                   FROM s);
```

(a) 1

(b) 3

(c) 5

(d) 10

(b)

8. What is the purpose of a cursor object when working with databases in Python?

(a) Create and manage database connections.

(b) Store tables from the database in memory for fast access.

(c) Execute SQL statements in the database.

(d) Define the database schema.

(e) Iterate over the rows of a query result.

(c) and (e)

2 NumPy (18 points)

Write a function `is_latin_square(matrix)` that takes a 2D NumPy array `matrix` representing a square matrix of integers. The function should return `True` if the matrix is a **Latin square**, and `False` otherwise.

A **Latin square** of size $n \times n$ is a grid filled with the numbers $1, 2, \ldots, n$ such that each number appears exactly once in each row and exactly once in each column.

For example, the following is a Latin square of size 4×4:

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \\ 3 & 4 & 1 & 2 \\ 4 & 1 & 2 & 3 \end{pmatrix}$$

Do not use explicit loops to check the rows and columns. Use only NumPy's array operations and functions for an efficient solution.

Solution:

```
def is_latin_square(matrix):
    n = matrix.shape[0]

    # Create a range of values 1 to n
    seq = np.arange(1, n + 1)

    # Check if all rows contain the values 1 to n
    rows_check = np.all(np.sort(matrix, axis=0) == seq.reshape(-1, 1))

    # Check if all columns contain the values 1 to n
    cols_check = np.all(np.sort(matrix, axis=1) == seq)

    return rows_check and cols_check
```

3 Pandas (20 points)

The following DataFrame contains data on basketball college players.

	Name	Team	Position	Number	Age	Height	Weight	Salary
0	Jay	Boston Eagles	PG	3	33	6.2	89	100000
1	Peter	Boston Eagles	SF	4	25	6.4	79	120000
2	James	Columbia Lions	SG	7	34	5.9	113	95000
3	Jack	Princeton Tigers	SF	11	35	6.1	78	115000
4	Steven	Columbia Lions	SG	5	28	5.8	84	150000
5	Mike	Princeton Tigers	C	12	27	6.9	102	135000
6	Robert	Boston Eagles	PG	6	30	6.1	90	125000
7	Chris	Columbia Lions	PF	9	26	6.6	87	140000

Assume that the DataFrame is stored in a variable called `df`. Using this DataFrame, write the following queries (4 points for each query):

1. Display the name and age of players who play for 'Boston Eagles' and their age is above 28.

```
df[(df['Team'] == 'Boston Eagles') & (df['Age'] > 28)][['Name', 'Age']]
```

2. Display the name and height of the tallest player.

```
df.loc[df['Height'].idxmax(), ['Name', 'Height']]
```

3. Show the average salary of players in each team.

```
df.groupby('Team')['Salary'].mean()
```

4. Add a new column BMI to the DataFrame, calculated using the formula:

$$BMI = \frac{\text{Weight (kg)}}{(\text{Height (m)})^2}$$

Assume that height is in feet and weight is in kilograms (1 foot = 0.3048 meters).

```
df['BMI'] = df['Weight'] / ((df['Height'] * 0.3048) ** 2)
```

5. Find players whose salary is above the average salary in their team. Show their name, team, and salary along with the average salary of their team.

```
df['Avg_Team_Salary'] = df.groupby(['Team'])['Salary'].transform('mean')
df[df['Salary'] > df['Avg_Team_Salary']][['Name','Team','Salary','Avg_Team_Salary']]
```

4 SQL (30 points)

You have a database for a coaching club with the following tables (underlined attributes represent the primary key):

```
coach(coach_id, name, e_mail, from_date, hourly_rate)
types(type_name, description)
coaches(coach_id, type_name)
clients(client_id, address, mobile)
training_program(client_id, start_date, coach_id, type_name, hours)
```

Description of the tables:

- The table `coach` contains data on coaches that work in the club. Each coach has an id, name, e-mail address, date of starting his/her job as a coach (`from_date`) and hourly rate.
- The table `types` contains data on the coaching types offered in the club, including the type name (e.g., "life coaching", "career coaching", etc.) and description.
- The table `coaches` describes which coaching types are offered by each coach. Each coaching type has at least one coach who offers it.
- The table `clients` contains data on clients of the club. Each client has an id, name, address and mobile phone. Each client has at least one training program.
- The table `training_program` contains data on the training programs of the clients. For each client, it stores the starting date of the program (`start_date`), id of the coach, the coaching type (`type_name`), and the number of total hours planned for the program. The cost of the program is calculated based on the number of hours and the hourly rate of the selected coach.

Write the following queries in SQL (5 points for each query):

1. Find all the clients that have been trained in a coaching type whose description contains the word "life".

```sql
SELECT tp.client_id
FROM training_program AS tp
JOIN types AS t
ON tp.type_name = t.type_name
WHERE t.description LIKE '%life%';
```

2. Find pairs of different clients who started their training at the same day with the same coach. The result columns should include the ids of the customers and the starting date of the training. Each such pair should appear only once in the result.

```sql
SELECT t1.client_id, t2.client_id, t1.start_date
FROM training_program AS t1
JOIN training_program AS t2
ON t1.client_id < t2.client_id
WHERE t1.start_date = t2.start_date AND t1.coach-id = t2.coach-id;
```

3. Find all the clients who have never been trained by a coach named Levi.

```sql
SELECT client_id
FROM clients
WHERE client_id NOT IN (
    SELECT tp.client_id
    FROM training_program AS tp
    JOIN coach AS c
    ON tp.coach_id = c.coach_id
    WHERE c.name = 'Levi'
);
```

4. Find for each client the total amount he/she has to pay for all his/her training programs.

```sql
SELECT tp.client_id, SUM(tp.hours * c.hourly_rate) AS total_pay
FROM training_program AS tp
JOIN coach AS c
ON tp.coach_id = c.coach_id
GROUP BY tp.client_id;
```

5. Find customers who have been trained in **all** the coaching types offered by the club.

```sql
SELECT c.client_id
FROM clients AS c
WHERE NOT EXISTS (
    SELECT *
    FROM types
    WHERE type_name NOT IN (
        SELECT tp.type_name
        FROM training_program AS tp
        WHERE tp.client_id = c.client_id
    )
);
```

6. Find customers who have had at least 3 different training programs, and all their training programs have been carried out by the same coach.

```sql
SELECT client_id
FROM training_program
WHERE client_id NOT IN (
    SELECT t1.client_id
    FROM training_program AS t1
    JOIN training_program AS t2
    ON t1.client_id = t2.client_id
    WHERE t1.coach_id <> t2.coach_id
)
GROUP BY client_id
HAVING COUNT(*) >= 3;
```