Data 101/Info 258: Data Engineering
Midterm Exam Solutions

UC Berkeley, Spring 2025

March 12, 2025

Name: ______________________________

Email: ______________________________@berkeley.edu

Student ID: ______________________________

Examination room: ______________________________

Name of the student on your left: ______________________________

Name of the student on your right: ______________________________

Instructions

Do not open the examination until you are instructed to do so.

This exam consists of **101** ~~97~~ points spread over **5 questions** (including the Honor Code), and must be completed in the 110-minute time period on March 12, 2025, 6:10pm – 8:00pm unless you have pre-approved accommodations otherwise.

For multiple-choice questions, select **one choice** for circular bubble options, and select **all choices that apply** for box bubble options. In either case, please indicate your answer(s) by **fully** shading in the corresponding circle/box.

Make sure to write your SID on each page to ensure that your exam is graded.

Honor Code **[1 pt]**

As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others. I am the person whose name is on the exam, and I completed this exam in accordance with the Honor Code.

Signature: ______________________________

Chapter 1: Game Chat [30 pt]

You are a new data engineer at an AI company that tries to detect toxic and positive messages sent in game chats, in order to promote a healthy gaming environment. As the company grows, it has gained many clients (game studios) who use the company's service.

The company stores information in the following relations.

```
games (id, name, type, studio)
players (id, name, game_id, reputation_score, signup_date, status)
messages (id, game_id, player_id, message, created_at)
```

Games Table (games)

```
CREATE TABLE games (
    id INT,
    name VARCHAR(30) NOT NULL,
    type TEXT,
    studio TEXT
);
```

Players Table (players)

```
CREATE TABLE players (
    id INT,
    name VARCHAR(30) NOT NULL,
    game_id INT,
    reputation_score INT,
    signup_date DATE,
    status TEXT
);
```

Messages Table (messages)

```
CREATE TABLE messages (
    id INT,
    game_id INT,
    player_id INT,
    message TEXT,
    created_at DATE
);
```

1.1. You realize that your company was “moving fast" with its data models early on, so you need to fix things before they get worse.

You start with the basics, such as what constraints may be present in the data, so that your relations are set up right. You start by inspecting the relations.

i. [3 pt] Which columns could be primary keys? (Select all that apply.)

☐ **A. games.id**

☐ B. games.type

☐ **C. players.id**

☐ D. players.game_id

☐ E. players.status

☐ **F. messages.id**

☐ G. messages.game_id

☐ H. messages.player_id

ii. [3 pt] Which columns could be foreign keys? (Select all that apply.)

☐ A. games.id

☐ B. games.type

☐ C. players.id

☐ **D. players.game_id**

☐ E. players.status

☐ F. messages.id

☐ **G. messages.game_id**

☐ **H. messages.player_id**

iii. [3 pt] Games using our service are doing very well; they have gained many new players! But the games (your clients) want to impose a **constraint**: they want to ensure that when a new player signs up, they can't reuse an existing player name.

Note that we don't want to prevent users from reusing their player name across games. But per game, we need to ensure that the player name is only used once. What kind of **constraint** should we implement to the players table? On what **column(s)**? Fill in the blanks below.

```
ALTER TABLE players ADD CONSTRAINT new_constraint _____________
(__________________________);
```

Solution: ALTER TABLE players ADD

CONSTRAINT new_constraint UNIQUE (game_id, player_name);

1.2. i. [4 pt] Create a view called `positive_messages` that retrieves all messages containing the phrases "good game" and "thank you" sent in the year 2024. These messages are used to assess the overall positivity of user interactions in each game. Note that all messages are already in **lower case**.

Example Output:

id	game_id	player_id	message	created_at
18927	01	13678	'that was a good game'	'2024-12-31'
18716	07	58493	'thank you :)'	'2024-12-31'
17938	03	47361	'awww thank you!'	'2024-12-31'
			......	
16119	04	01829	'good game guys'	'2024-01-01'
16103	02	36720	'bruhhhhh good game'	'2024-01-01'

Available relations (repeated here for your convenience):

```
games (id, name, type, studio)
players (id, name, game_id, reputation_score, signup_date, status)
messages (id, game_id, player_id, message, created_at)
```

```
CREATE VIEW positive_messages AS

SELECT *

FROM ______________________________

______________________________

______________________________

______________________________

______________________________;
```

Solution:

```
CREATE VIEW positive_messages AS
SELECT *
FROM messages
WHERE (message LIKE '%good game%'
OR message LIKE '%thank you%')
AND created_at BETWEEN '2024-01-01' AND '2024-12-31'
;
```

ii. [6 pt] Assuming your view is correct, fill in the blanks in the query below to calculate the positive message percentage, i.e., `positive_percentage`, defined as $\frac{\text{number of positive messages}}{\text{number of messages}}$, of each game in 2024, and return the top 5 games with the greatest `positive_percentage` in 2024.

Notes:

- Not all games may contain positive messages, but we know for certain that all games contain > 1 message.
- When calculating a percentage with division, make sure at least one of the numerator or denominator are floats, or else integer division will be performed (i.e., $\frac{1}{2.0} = 0.5$, but $\frac{1}{2} = 0$ with integer division).

Example Output:

game_id	game_name	positive_percentage
08	'Cal Hunter'	24.37
02	'Cpex'	21.70
04	'The Legend of Oski'	18.69
01	'League of Hackers'	12.01
10	'Berkeley Impact'	9.40

Available relations (repeated here for your convenience):

```
games (id, name, type, studio)
players (id, name, game_id, reputation_score, signup_date, status)
messages (id, game_id, player_id, message, created_at)
```

```
SELECT g.id AS game_id, g.name AS game_name,
________________________________________
________________________________________
________________________________________
________________________________________
________________________________________
AS positive_percentage
FROM games AS g
LEFT JOIN positive_messages AS p
ON g.id = p.game_id
________________________________________
________________________________________
________________________________________
;
```

Solution:

```
SELECT g.id AS game_id, g.name AS game_name,
 FLOAT(COUNT(p.id))/(
    SELECT FLOAT(COUNT(*))
    FROM messages
    WHERE created_at BETWEEN '2024-01-01' AND '2024-12-31'
    AND game_id = g.id
    ) * 100 AS positive_percentage
FROM games AS g
LEFT JOIN positive_messages AS p
ON g.id = p.game_id
GROUP BY g.id, g.name
ORDER BY positive_percentage DESC
```

```
LIMIT 5;
```

iii. [6 pt] A player will be awarded an increase to their reputation score when they send a positive message, and their reputation score will be deducted if they send a negative message.

To promote a friendly gaming environment, the game "Cal Hunter" wants to create an honor list to award players that have the highest reputation score. Write a SQL query to create an honor list for the game "Cal Hunter". If two (or more) players have the same reputation score, **they get the same rank**.

The top five rows of the output table should look like this:

player_id	player_name	reputation_score	rank
18390	'Zack101'	1980	1
01350	'LiZ'	1975	2
33267	'immichelle'	1975	2
20209	'cx:)'	1892	3
28394	'WESLEY233'	1840	4

Available relations (repeated here for your convenience):

```
games (id, name, type, studio)
players (id, name, game_id, reputation_score, signup_date, status)
messages (id, game_id, player_id, message, created_at)
```

```
SELECT ______________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________
FROM ________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________
_____________________________________________________;
```

Solution:

```
SELECT p.player_id,
 p.player_name,
 p.reputation_score,
 RANK() OVER (ORDER BY p.reputation_score DESC) AS rank
FROM players AS p
INNER JOIN games AS g
ON p.game_id = g.id
WHERE g.name = 'Cal Hunter'
ORDER BY rank
;
```

iv. [5 pt] A player will be banned from a game if their reputation score is lower than 50. There's an anonymous report submitted by a player claiming that multiple players are being banned by mistake even though their reputation score is not lower than 50. Fill in the blanks below to find all games that have players with reputation scores higher or equal to 50 while their status is "banned". Use a subquery as indicated and fill in the blanks.

Available relations (repeated here for your convenience):

```
games (id, name, type, studio)
players (id, name, game_id, reputation_score, signup_date, status)
messages (id, game_id, player_id, message, created_at)
```

```
SELECT id, name
____________________________________________
WHERE ______________________________________(
____________________________________________
____________________________________________
____________________________________________
____________________________________________
__________________________________________);
```

Solution:

```
SELECT id, name
FROM games AS g
WHERE EXISTS (
 SELECT *
 FROM players AS p
 WHERE p.game_id = g.id
 AND p.status = 'banned'
 AND p.reputation_score >= 50
);
```

Chapter 2: Food Delivery Order Management System [20 pt]

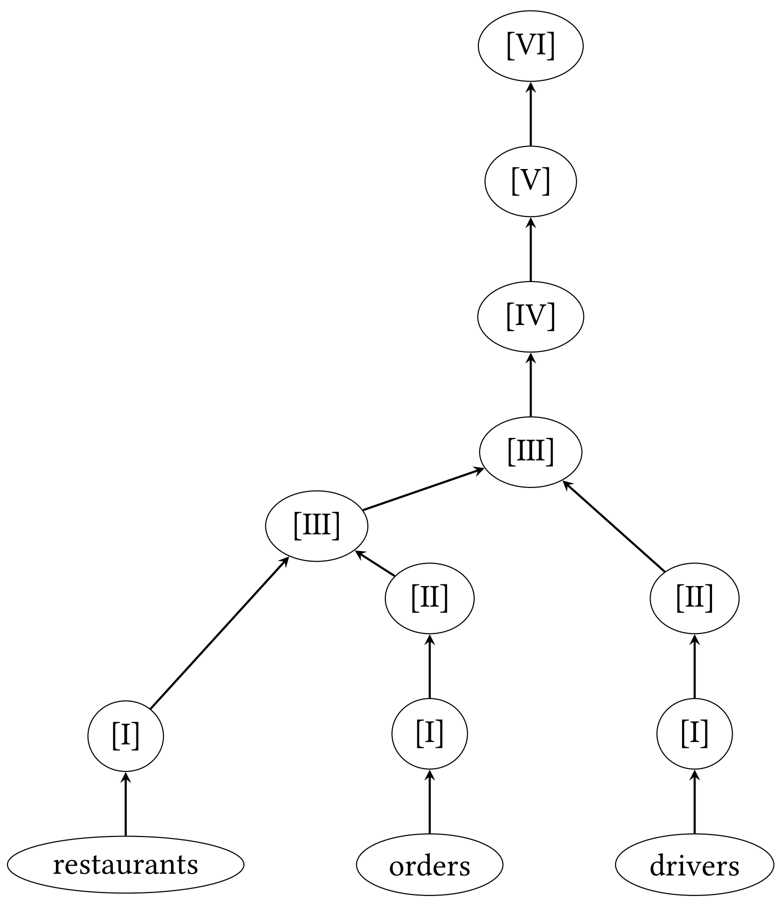
Consider a simplified version of a food delivery platform, where users place orders from restaurants, drivers deliver the orders, and the platform tracks the details of each transaction. Below is the relevant part of the database schema:

```
orders (oid, rid, did, amount, platform)
restaurants (rid, name, cuisine, minimum_amount)
drivers (did, name, rating, vehicle)
```

2.1. [6 pt] Suppose we write the below query:

```
SELECT r.name, SUM(o.amount) FROM restaurants r, orders o, drivers d
WHERE r.rid = o.rid AND o.did = d.did
    AND o.platform = 'UberEats' AND d.rating >= 4.5
GROUP BY r.name HAVING COUNT(*) > 100;
```

The query optimizer then produces the following execution plan, according to SQL query semantics. Based on the execution plan, answer the question below.



What extended relational operators should be in the nodes marked [I], [II], [III], [IV], [V], and [VI]?

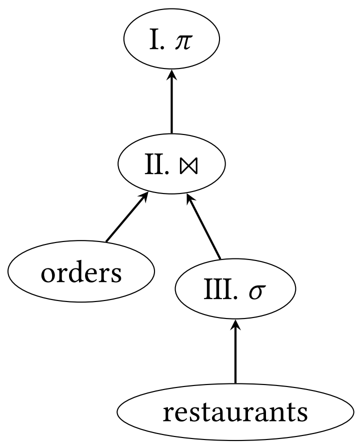
[I]	◯ π	◯ ρ	◯ σ	◯ $\bowtie$	◯ $\times$	◯ γ
[II]	◯ π	◯ ρ	◯ σ	◯ $\bowtie$	◯ $\times$	◯ γ
[III]	◯ π	◯ ρ	◯ σ	◯ $\bowtie$	◯ $\times$	◯ γ
[IV]	◯ π	◯ ρ	◯ σ	◯ $\bowtie$	◯ $\times$	◯ γ
[V]	◯ π	◯ ρ	◯ σ	◯ $\bowtie$	◯ $\times$	◯ γ
[VI]	◯ π	◯ ρ	◯ σ	◯ $\bowtie$	◯ $\times$	◯ γ

2.2. [4 pt] Suppose we write the query below:

```
SELECT o.oid, r.name, o.amount
FROM orders o, restaurants r
WHERE o.rid = r.rid AND o.amount > r.minimum_amount AND o.amount = 50;
```

We rejigged the query optimizer to use some newfangled AI magic, so it now tries its level best to identify new execution plans with low cost.

The query optimizer produces the following execution plan.



For each relational operator, **write its corresponding subscript** according to the original SQL query, e.g. join conditions, selected attributes, etc. If there are no subscripts that could apply, write N/A.

Hint: Since we're using a newfangled AI query optimizer, the subscripts may go beyond clauses directly mentioned in the query directly—they could be derived from it.

[I] `o.oid, r.name, o.amount`

[II]

$o.rid = r.rid \land o.amount = 50$

[III] `50 > minimum_amount`

2.3. Select all relational algebra expressions that satisfy each description. For the queries below, to make them fit in one line, we will abbreviate restaurants to rest, orders to ord, and drivers to dri .

i. [3 pt] Get the names of restaurants that do not have any orders placed on DoorDash.

☐ **A.** $\pi_{\text{name}}(\text{rest}) - \pi_{\text{name}}\left(\text{rest} \bowtie_{\text{rid = ord.rid}} \sigma_{\text{platform='DoorDash'}}(\text{ord})\right)$

☐ B. $\pi_{\text{name}}\left(\text{rest} - \pi_{\text{name}}(\text{rest} \bowtie_{\text{rid = ord.rid}} \sigma_{\text{platform='DoorDash'}}(\text{ord}))\right)$

☐ C. $\pi_{\text{name}}\left(\sigma_{\text{platform}\neq\text{'DoorDash'}}\left(\text{rest} \bowtie_{\text{rid = ord.rid}} \text{ord}\right)\right)$

☐ D. $\pi_{\text{name}}\left(\sigma_{\text{platform='DoorDash'}}\left(\text{rest} \bowtie_{\text{rid = ord.rid}} \text{ord}\right)\right)$

☐ E. None of the above

ii. [3 pt] Get the ids of drivers who have completed more than 20 orders and have a vehicle type of either Toyota or Honda.

☐ A. $\pi_{\text{did}}\left(\sigma_{\text{COUNT}>20\wedge(\text{vehicle='Toyota'} \vee \text{vehicle='Honda'})}(\gamma_{\text{did,COUNT(*)}}(\text{ord} \bowtie_{\text{did=dri.did}} \text{dri}))\right)$

☐ **B.** $\pi_{\text{did}}\left(\sigma_{\text{COUNT}>20}(\gamma_{\text{did,COUNT(*)}}(\sigma_{\text{vehicle='Toyota'} \vee \text{vehicle='Honda'}}(\text{ord} \bowtie_{\text{did=dri.did}} \text{dri})))\right)$

☐ C. $\pi_{\text{did}}\left(\sigma_{\text{COUNT}>20}\left(\gamma_{\text{did,COUNT(*)}}\left(\text{ord} \bowtie_{\text{did=dri.did}} \text{dri}\right)\right)\right)$

☐ D. $\pi_{\text{did}}\left(\sigma_{\text{COUNT}>20\vee(\text{vehicle='Toyota'} \vee \text{vehicle='Honda'})}\left(\gamma_{\text{did,COUNT(*)}}\left(\text{ord} \bowtie_{\text{did=dri.did}} \text{dri}\right)\right)\right)$

☐ E. None of the above

2.4. [4 pt] **Extra Credit**: This is a challenging question. We encourage you to attempt it after completing the other parts of the exam!

Without using extended relational algebra operators (so no group by operator), write the following query in vanilla relational algebra:

Find the name of the restaurant(s) with the lowest target minimum amount. If there are multiple, return all of them.

Solution: $\pi_{\text{name}}(\text{rest}) - \pi_{\text{r1.name}}\left(\sigma_{\text{r1.minimum\_amount}>\text{r2.minimum\_amount}}\left(\rho_{\text{r1}}(\text{rest}) \times \rho_{\text{r2}}(\text{rest})\right)\right)$

Chapter 3: Staff Social! [10 pt]

This week, course staff is organizing a staff social and is collecting social ideas from its members. Each staff member may voluntarily suggest any number of ideas, each of which is recorded in the `ideas` table. Information about staff members is stored in the `staff` table. You may assume you can see the full tables below.

Staff Table (`staff`)

```
CREATE TABLE staff (
    id INT PRIMARY KEY,
    name VARCHAR(50) NOT NULL,
    team TEXT
);
```

id	name	team
100	Zack	Infra
101	Aditi	Grading
102	Nehal	Tutoring
103	Michelle	Course Notes
104	Manas	Tutoring

Ideas Table (`ideas`)

```
CREATE TABLE ideas (
    idea_id INT PRIMARY KEY,
    date DATE,
    time TIME,
    location VARCHAR(50) NOT NULL,
    member_id INT,
    description TEXT,
    FOREIGN KEY (member_id) REFERENCES staff(id),
    UNIQUE (date, time)
);
```

idea_id	date	time	location	member_id	description
1	2025-03-12	14:00	SF	104	Golden Gate Park
2	2025-03-12	18:00	Glade	101	Picnic and games
3	2025-03-13	12:00	Berkeley Marina	103	NULL
4	2025-03-14	15:30	TP Tea	102	Boba!

3.1. [2 pt] Which of the following changes will ALWAYS execute successfully (will not produce an error)?

☐ A. Deleting a tuple from the `staff` table

☐ **B. Deleting a tuple from the `ideas` table**

☐ C. Altering the `member_id` field of an existing tuple in the `ideas` table

☐ **D. Setting the `time` field of an existing tuple in the `ideas` table to NULL**

3.2. [2 pt] Which of the following lines of code would produce a constraint violation of any kind?

☐ A. `INSERT INTO staff VALUES (105, 'Aldrin', 'Tutoring');`

☐ **B. `INSERT INTO staff VALUES (101, 'Li', 'Projects');`**

☐ C. `INSERT INTO staff VALUES (106, 'Chenxi');`

☐ **D. `INSERT INTO ideas VALUES (5, '2025-03-15', '16:00', 'Big C', 110, 'Hiking trip!');`**

☐ E. `INSERT INTO ideas VALUES (6, '2025-03-16', '11:00', 'BAMPFA', 101, NULL);`

> **Solution:** Choice B inserts a duplicate primary key and Choice D has an invalid foreign key reference.

3.3. [2 pt] As a safeguard, Christy wants to ensure that should a staff member be deleted from the `staff` table, all of their proposed ideas are deleted from the `ideas` table. Select which table schema she should modify and write out the modification below.

○ A. `staff`

○ **B. `ideas`**

____

> **Solution:** ON DELETE CASCADE

3.4. [2 pt] Wesley proposes implementing a voting system to measure the popularity of each social idea. He considers two different approaches:

1. Modify the schema of the `ideas` table by adding a `votes` column, storing the vote count.
2. Create a separate `votes` table with schema: `(member_id INT, idea_id INT, comments TEXT, PRIMARY KEY (member_id, idea_id))`.

Which approach is preferable? Justify your answer in two sentences or less.

Solution: Possible answers:

- Option 1 is simpler because the vote count for each idea is stored directly in the `ideas` table. However, updating the vote count requires an expensive `UPDATE` operation.
- Option 2 is preferable because it avoids modifying the existing `ideas` schema and ensures each staff member can only vote once per idea using a primary key constraint.

3.5. [2 pt] Based on your choice above, write out the appropriate SQL modification.

Solution: Possible answers:

- `ALTER TABLE ideas ADD COLUMN votes INT DEFAULT 0;`
- `CREATE TABLE votes (member_id INT NOT NULL, idea_id INT NOT NULL, comments TEXT, PRIMARY KEY (member_id, idea_id), FOREIGN KEY (member_id) REFERENCES staff(id), FOREIGN KEY (idea_id) REFERENCES ideas(idea_id));`

Chapter 4: RSF IM Teams! [15 pt]

Data 101 got access to the RSF's (Recreational Sports Facility's) IM sports teams database! Let's take a look at what they have here…

```
CREATE TABLE users (
    user_id INT PRIMARY KEY,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    team_id INT,
    FOREIGN KEY (team_id) REFERENCES Teams(team_id)
);

CREATE TABLE teams (
    team_id INT PRIMARY KEY,
    team_name VARCHAR(100) UNIQUE NOT NULL,
    captain_id INT,
    sport_type VARCHAR(50) NOT NULL,
    FOREIGN KEY (captain_id) REFERENCES Users(user_id)
);

CREATE TABLE matches (
    match_id INT PRIMARY KEY,
    team1_id INT,
    team2_id INT,
    match_date DATE NOT NULL,
    start_time TIME NOT NULL,
    end_time TIME,
    winner_team_id INT,
    FOREIGN KEY (team1_id) REFERENCES Teams(team_id),
    FOREIGN KEY (team2_id) REFERENCES Teams(team_id)
);
```

In summary, we have the following tables:

`users(user_id, first_name, last_name, email, team_id)`,

`teams(team_id, team_name, captain_id, sport_type)`,

`matches(match_id, team1_id, team2_id, match_date, start_time, end_time, winner_team_id)`.

4.1. [2 pt] Which query plan is it plausible for the query below to follow? Choose the correct option.

```
SELECT * FROM users WHERE team_id >= 5 AND team_id < 10;
```

◯ A.

```
Limit  (cost=0.14..20.64 rows=1 width=802)
        (actual time=0.005..0.009 rows=0 loops=1)
   ->  Nested Loop  (cost=0.14..20.64 rows=1 width=802)
                    (actual time=0.004..0.005 rows=0 loops=1)
         ->  Seq Scan on users  (cost=0.00..12.40 rows=1 width=462)
                                (actual time=0.003..0.004 rows=0 loops=1)
               Filter: ((team_id >= 5) AND (team_id < 10))
         ->  Index Scan using teams_pkey on teams
                            (cost=0.14..8.16 rows=1 width=344) (never executed)
               Index Cond: (team_id = users.team_id)
 Planning Time: 0.668 ms                    Execution Time: 0.055 ms
```

◯ **B.**

```
Seq Scan on users  (cost=0.00..12.40 rows=1 width=462)
                    (actual time=0.002..0.004 rows=0 loops=1)
   Filter: ((team_id >= 5) AND (team_id < 10))
 Planning Time: 0.244 ms                    Execution Time: 0.043 ms
```

◯ C.

```
Index Scan using teams_pkey on teams  (cost=0.14..8.16 rows=1 width=344)
                                      (actual time=0.006..0.009 rows=0 loops=1)
   Index Cond: ((team_id >= 5) AND (team_id < 10))
 Planning Time: 0.281 ms                    Execution Time: 0.043 ms
```

◯ D.

```
Index Scan using teams_pkey on teams  (cost=0.14..8.16 rows=1 width=344)
                                      (actual time=0.006..0.009 rows=0 loops=1)
   Index Cond: ((team_id >= 5) AND (team_id < 10))
 Planning Time: 0.281 ms                    Execution Time: 0.043 ms
```

◯ E. None of the Above.

4.2. For each query below, select which join the query optimizer is most likely to perform. Note that for this question, consider the `users` table to be small, while the `teams` and `matches` tables are large.

i. [1 pt]
```
SELECT *
FROM matches, teams
WHERE teams.team_id = matches.team2_id
ORDER BY teams.team_name
```

◯ A. Hash Join

◯ **B. Sort Merge Join**

○ C. Nested Loop Join

ii. [1 pt]
```
SELECT *
FROM matches, teams
WHERE teams.team_id < matches.team1_id
AND teams.team_id > matches.team2_id
LIMIT 100
```

○ A. Hash Join

○ B. Sort Merge Join

○ **C. Nested Loop Join**

iii. [1 pt]
```
SELECT * FROM users, teams
WHERE teams.team_id = users.team_id
```

○ **A. Hash Join**

○ B. Sort Merge Join

○ C. Nested Loop Join

4.3. [2 pt] Consider the following query:

```
SELECT u.first_name, u.last_name, t.team_name, m.match_date
FROM users u
JOIN teams t ON u.team_id = t.team_id
JOIN matches m ON t.team_id = m.team1_id OR t.team_id = m.team2_id
WHERE m.match_date >= CURRENT_DATE AND m.winner_team_id = t.team_id
```

Note: `CURRENT_DATE` just returns the current date in the default timezone of the database.

What are viable optimizations that can be made (check all that apply)? Note that "pushdown" of some X, "past" some Y, refers to making the X operation happen before the Y operation.

☐ **A. Selection pushdown of `m.match_date >= CURRENT_DATE` past the `JOIN` involving the `matches` table, to decrease the number of rows before joining**

☐ B. To reduce columns at each step, push down projections all the way down to each respective table: [`u.first_name, u.last_name`] to `users`, `t.team_name` to `teams`, and `m.match_date` to `matches`.

☐ C. Selection pushdown of `m.winner_team_id = t.team_id` past all the joins, to decrease the number of rows before joining.

☐ D. None of these are valid optimizations

4.4. [6 pt] Let's do some cardinality estimation to figure out the right order of joins.

For this question only, assume there are:

- 5000 tuples in the `users` table

- 500 tuples in `teams` table
- 1000 tuples in the `matches` table

i. If you were to right join the users table onto the teams table on the team_id columns (i.e., `FROM users AS u RIGHT JOIN teams AS t ON u.team_id = t.team_id`), at most how many rows will be output? Assume at most 10% of the registered teams are old, invalid teams—these teams don't have any of their members as listed users in the RSF database.

5050

ii. If you were to inner join the users table with itself, on team_id (i.e., `FROM users AS u1 INNER JOIN users AS u2 ON u1.team_id = u2.team_id`), at most, how many output rows would you get? For only this question, assume that each team has a minimum of 4 members and a maximum of 10 members in the users table. You can put a number, or N/A if there is not enough information.

50,000

iii. If you need to natural join all three tables (users, teams, and matches), which join order is the most efficient (keeping in mind, minimizing the number of rows)?

◯ **A. Join `users` and `teams` first, then join `matches`**

◯ B. Join `teams` and `matches` first, then join `users`

◯ C. Join `users` and `matches` first, then join `teams`

◯ D. Join order doesn't matter

4.5. [2 pt] For the following query, which of the answer choices below will speed up the query (in comparison to not doing anything)? Select all that apply.

```
SELECT u.first_name, u.last_name
FROM users u
WHERE u.team_id > 5 OR u.last_name LIKE 'A%'
```

☐ A. Building index on only `team_id` or only `last_name`

☐ **B. Building separate indexes on `team_id` and `last_name`**

☐ C. Building a multi-column index on columns `team_id` and `last_name`

☐ D. Building an index would not speed up the execution of this query

Chapter 5: NBA 2K25? [20 pt]

One of the games analyzed from *Game Chat* (remember Chapter 1?) is a spin-off of NBA 2K25. After the recent NBA trade deadline events, some course staff members ran a tournament where they played a couple of games, with each match featuring staff members controlling NBA players in 5v5 games. Staff wants to analyze performance across different roles (**Playmaker, Shooter, Defender**) and improve game strategies.

`game_performance`: Tracks individual performance per game

```
CREATE TABLE game_performance (
    game_id INT,
    staff_id INT,
    staff_name VARCHAR(50),
    points INT,
    assists INT,
    rebounds INT
);
```

game_id	staff_id	staff_name	nba_player	points	assists	rebounds
1	101	Christy	Stephen Curry	26	10	5
1	102	Zackary	Luka Doncic	45	13	11
			......			
2	101	Christy	Stephen Curry	54	6	4
2	102	Zackary	Kevin Durant	32	8	6

5.1. [2 pt] Assume there are 1000 rows in the dataset with 100 unique staff members. The columns have the following storage costs per entry on average:

- `INTEGER` is stored as 4 bytes
- `VARCHAR(50)` is stored as 10 bytes

Compute the total storage cost required to store both columns (**staff_id**, **staff_name**) in the `game_performance` table in bytes. Make sure to show your work.

________ B

Solution:

`staff_id` (4 bytes per row): $1000 \times 4 = 4000$ bytes

`staff_name` (10 bytes per row): $1000 \times 10 = 10000$ bytes

Total storage cost before encoding: $4000 + 10000 =$ **14000** bytes

5.2. [2 pt] To encode `staff_name` efficiently, we decided to use a fixed binary encoding where each unique staff name is assigned a $\log_2(100)$-bit binary code instead of storing the full string.

i. How many bits are required to store a single `staff_name` under this encoding?

________bits

Solution:

$$\log_2(100) \approx 6.64 \text{ bits} \approx 7 \text{ bits}$$

ii. Compute the total storage cost if we replace `staff_name` with its binary code representation.

________bytes

Solution:

New storage cost for `staff_name`: 1000×7 bits $= 7000$ bits $= 875$ bytes

New total storage cost (with encoding): $4000 + 875 = \mathbf{4875}$ bytes

5.3. [1 pt] Determine whether replacing `staff_name` with its binary encoding minimizes the total encoding cost.

- **A. True**
- B. False

5.4. [2 pt] If the number of unique staff members increases to 500, but the total rows remain 1000, how does this impact the effectiveness of binary encoding under MDL assuming that the number of staff members will continually increase? Explain your answer using a maximum of three sentences.

- A. Increase in effectiveness
- **B. Decrease in effectiveness**
- C. No impact in effectiveness

Solution: With 500 unique names, the binary encoding will increase the total storage cost for `staff_name` from 875 bytes to 1125 bytes. This means that the efficiency of the encoding has decreased because we now need more bits to represent each unique name.

5.5. [6 pt] Using the tables provided below (rows omitted), write a SQL query to calculate each staff member's ranking *per game* based on `points` scored, using the `RANK()` window function. You may not need to use all the space/lines provided.

`game_performance`: Tracks individual performance per game

game_id	staff_id	staff_name	nba_player	points	assists	rebounds
1	101	Christy	Stephen Curry	26	10	5
1	102	Zackary	Luka Doncic	45	13	11
			......			
2	101	Christy	Stephen Curry	54	6	4
2	102	Zackary	Kevin Durant	32	8	6

`game_details`: Tracks metadata about each game

game_id	game_date	opponent_team	home_away
1	2024-01-05	Lakers	Home
		......	
2	2024-01-10	Suns	Away

Example Output:

game_id	staff_id	staff_name	nba_player	points	rank
1	101	Christy	Stephen Curry	26	2
1	102	Zackary	Luka Doncic	45	1
			......		
2	101	Christy	Stephen Curry	54	1
2	102	Zackary	Kevin Durant	32	2

```
SELECT ______________________________________________

________________________________________________

__________________________________________AS rank_in_game

FROM ________________________________________________

ORDER BY ____________________________________________;
```

Solution:

```
SELECT game_id,
       staff_id,
       staff_name,
       nba_player,
       points,
       RANK() OVER (PARTITION BY game_id ORDER BY points DESC) AS rank_in_game
FROM game_performance
ORDER BY game_id;
```

5.6. [7 pt] Recall that we also have access to two tables: `game_performance`, which tracks the individual performance of staff members in each game, and `game_details`, which contains metadata about the games, such as the date, opponent team, and home/away status.

Now, we want to determine the ranking of each staff member within their respective game based on `points` scored, using the `RANK()` function. Additionally, for staff members who have participated in multiple games, you need to compute the difference in rank compared to their previous game.

Example Output:

game_id	staff_id	staff_name	points	prev_rank	rank_in_game	rank_change
1	102	Zackary	45	1	1	NULL
1	101	Christy	26	2	2	NULL
			......			
2	101	Christy	54	2	1	1
2	102	Zackary	32	1	2	-1

Exam Correction: The skeleton code for IV should actually be for V, and VI should be swapped with V.

```
WITH ranked_performance AS (
 SELECT
  gp.game_id, gd.game_date,
  ______________(I)______________,
  gp.points,
  RANK() OVER (__________(II)__________) AS rank_in_game
FROM game_performance AS gp
JOIN ______________(III)______________
)

SELECT
 ______________(IV)______________,
 ____________(V)____________AS prev_rank,
 ______(V)______ - ______(VI)______AS rank_change
FROM ranked_performance AS rp
ORDER BY ______________(VII)______________;
```

I: ______________

II: ______________

III: ______________

IV: LAG(__________) OVER (______________)

V: ______________

VI: ______________

VII: ______________

Solution:

Note: The skeleton code for IV should actually be for V, and VI should be swapped with V.

```
WITH ranked_performance AS (
    SELECT
        gp.game_id,
        gd.game_date,
        gp.staff_id,
        gp.staff_name,
        gp.points,
        RANK() OVER (PARTITION BY gp.game_id ORDER BY gp.points DESC) AS rank_in_game
```

```
    FROM game_performance AS gp
    JOIN game_details AS gd ON gp.game_id = gd.game_id
)

SELECT
    rp.game_id,
    rp.staff_id,
    rp.staff_name,
    rp.points,
    LAG(rp.rank_in_game) OVER (PARTITION BY rp.staff_id ORDER BY rp.game_date)
        AS prev_rank,
    rp.rank_in_game,
    rp.rank_in_game - LAG(rp.rank_in_game) OVER (PARTITION BY rp.staff_id
        ORDER BY rp.game_date) AS rank_change
FROM ranked_performance AS rp
ORDER BY rp.staff_id;
```

Chapter 6: Congratulations! [0 pt]

Congratulations! You have completed this exam.

- Make sure that you have written your Student ID number on every other page of the exam. You may lose points on pages where you have not done so.
- Also ensure that you have signed the Honor Code on the cover page of the exam.
- If more than 10 minutes remain in the exam period, you may hand in the exam **and** the reference packet and leave.
- If $\leq$ 10 minutes remain, please sit quietly until the exam concludes.

[Optional, 0 pts] Use this page to draw your favorite Data 101/Info 258 moment!