Foster Mosden
Big Data Midterm

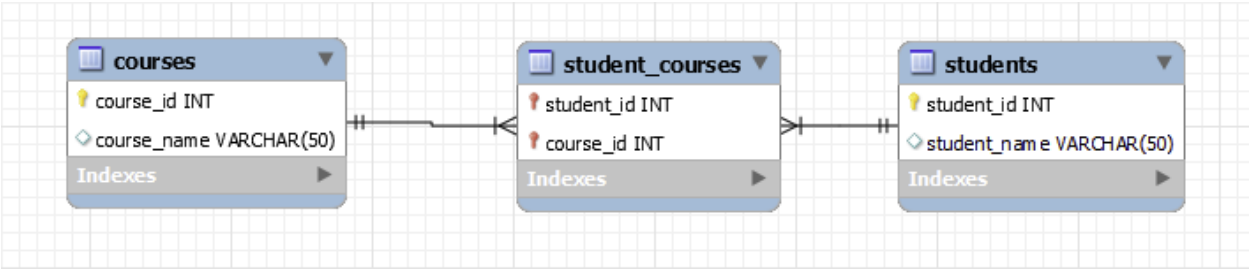
QUESTION 1:

1. Briefly describe the requirements of a 3rd normal form (3NF).

To be in 3NF, firstly, the table must adhere to the 1st Normal Form (1NF); each column must contain only atomic values, avoiding multiple values or sets. Secondly, it should satisfy the 2nd Normal Form (2NF) requirements; it should not have partial dependencies; all non-key attributes must be fully functionally dependent on the entire primary key. A table in a database is in 3rd Normal Form (3NF) if it is already in 2nd Normal Form (2NF) AND it eliminates transitive dependencies. 3NF aims to reduce redundancy and enhance data integrity.

2. Briefly describe M:N relationship between two entities (share an ER model)

An M:N (many-to-many) relationship in a database occurs when multiple records in one table are related to multiple records in another table. This is typically executed using an intermediate table called a junction table, which stores the primary keys from both tables, creating pairs of relationships. In the example ER model below, student_courses serves as our junction table. The tables `students` and `courses` both have a 1:N relationship with student_courses, effectively creating an M:N relationship, with `student_courses` simply acting as the key pairing middle man.



3. Briefly describe when a “WHERE” clause can and cannot be used to select rows after a “GROUP BY” clause?

The WHERE clause is used to filter rows before they are grouped using GROUP BY. This also means that the WHERE clause cannot be used to filter rows based on the result of aggregate functions, because the aggregate functions operate on the grouped rows. For such filtering, the HAVING clause would need to be used, after the GROUP BY clause.

4. Briefly discuss three differences between normalized modeling (OLTP database) and dimensional modeling (OLAP database).

Normalized Modeling (OLTP Database) is designed for transactional systems, as it is optimized for write operations. OLTP modeling reduces redundancy, and enforces data integrity through normalization. On the other hand, dimensional Modeling (OLAP Database) is designed for analytical processing and reporting, as it is optimized for read-heavy queries. OLAP modeling uses denormalized tables (such as in the star or snowflake schema) for faster query performance.

5. Briefly discuss two differences between the NoSQL document database and a NoSQL wide column database.

The NoSQL Document Database stores data in flexible, JSON-like documents, and supports nested structures and complex data types. A NoSQL Wide Column Database stores data across many columns rather than rows. This approach increases schema flexibility, as columns can vary as much as is necessary per row.

6. Briefly discuss what the following command returns when used on documentDB: db.restaurants.find({"address.location": location})

This command finds document(s) in the "restaurants" collection where the "location" field within the "address" subdocument matches the provided location, ‘location’.

7. Briefly discuss the difference between “scan” and “get” in HBase

“Scan” retrieves a range of rows from a table, while “get” retrieves a specific row from a table based on the row key.

8. Briefly discuss why NoSQL databases are not good for “join” and “group” operations?

NoSQL databases sacrifice complex querying capabilities, including joins and group operations, for scalability and performance. Joins, especially, can be resource-intensive and are avoided in NoSQL designs.

9. Briefly discuss the max size of HRegion you will select when designing a database for TikTok?

HBase recommends keeping HRegion sizes between 10 GB to 20 GB for optimal performance. However, in the context of a short form video app where rapid data ingestion and retrieval are essential, I will advise choosing a smaller HRegion size, on the lower end of the recommended scale. This will help to ensure efficient data distribution and faster query processing. A conservative approach would be to set the HRegion size to around 10 GB, allowing for better load balancing, easier data management, and improved responsiveness in handling user-generated video content.

QUESTION 2:

```
# 2.1. Check if a movie is in stock
SELECT inventory_id, store_id
FROM inventory
WHERE film_id = (
      SELECT film_id
      FROM film
      WHERE title = 'ACADEMY DINOSAUR'
)
AND inventory_id NOT IN (
      SELECT inventory_id
      FROM rental
      WHERE return_date IS NULL
);
```

inventory_id	store_id
1	1
2	1
3	1
4	1
5	2

```
# 2.2 Checkout the movie for customer "Dwayne Olvera"
INSERT INTO rental(rental_date, inventory_id, customer_id, staff_id)
SELECT NOW(), inventory.inventory_id, customer.customer_id, 1
FROM inventory
JOIN film ON inventory.film_id = film.film_id
JOIN customer ON customer.store_id = inventory.store_id
WHERE film.title = 'ACADEMY DINOSAUR'
AND customer.first_name = 'DWAYNE'
AND customer.last_name = 'OLVERA'
AND inventory.inventory_id NOT IN (
      SELECT inventory_id
      FROM rental
      WHERE return_date IS NULL
)
LIMIT 1;
```

```
# Q2.3 Collect rental payment, and add a row in the payment table
INSERT INTO payment (customer_id, staff_id, rental_id, amount,
payment_date, last_update)
VALUES (
      (SELECT customer_id
      FROM customer
      WHERE first_name = 'DWAYNE' AND last_name = 'OLVERA'),
      1,
      (SELECT rental_id
      FROM rental
      WHERE customer_id = (SELECT customer_id
                        FROM customer
                        WHERE first_name = 'DWAYNE' AND last_name =
'OLVERA')
      ORDER BY rental_date DESC
      LIMIT 1),
      (SELECT rental_rate
      FROM film
      WHERE film_id = (
      SELECT film_id
      FROM inventory
      WHERE inventory_id = 1)),
      NOW(), NOW()
);
```

```sql
# 2.4 Overdue Movie List
SELECT
      film.title AS movie_title,
      rental.rental_date AS rental_date,
      rental.return_date AS return_date,
      film.rental_duration AS rental_duration,
      DATEDIFF('2006-02-18',
            DATE_ADD(rental_date,
                  INTERVAL rental_duration DAY)) AS days_overdue
FROM
      rental
      JOIN
      inventory ON rental.inventory_id = inventory.inventory_id
      JOIN
      film ON inventory.film_id = film.film_id
WHERE
      return_date IS NULL
      AND rental_duration < DATEDIFF('2006-02-18', rental_date)
ORDER BY rental.rental_date DESC;
```

movie_title	rental_date	return_date	rental_duration	days_overdue
PEACH INNOCENT	2006-02-14 15:16:03	NULL	3	1
SUIT WALLS	2006-02-14 15:16:03	NULL	3	1
SONS INTERVIEW	2006-02-14 15:16:03	NULL	3	1
TITANIC BOONDOCK	2006-02-14 15:16:03	NULL	3	1
TITANIC BOONDOCK	2006-02-14 15:16:03	NULL	3	1

QUESTION 3:

```
# 3.1 Load Positive and Negative Words Data into Separate Tables:
CREATE TABLE pos_words (words VARCHAR(255) PRIMARY KEY);
LOAD DATA LOCAL INFILE 'C:\\path\\positive-words.txt' INTO TABLE pos_words
FIELDS TERMINATED BY '\n';
CREATE TABLE neg_words (words VARCHAR(255) PRIMARY KEY);
LOAD DATA LOCAL INFILE 'C:\\path\\negative-words.txt' INTO TABLE neg_words
FIELDS TERMINATED BY '\n';
```

```
# 3.2 Write MySQL User-Defined Functions:
DROP FUNCTION IF EXISTS `yelp`.`CountPositiveWords`;
DELIMITER //
CREATE FUNCTION CountPositiveWords(input_text LONGTEXT) RETURNS INT
DETERMINISTIC
BEGIN
      DECLARE word_count INT DEFAULT 0;
      DECLARE positive_word VARCHAR(255);
      DECLARE done INT DEFAULT 0;
      DECLARE positive_words_cursor CURSOR FOR
      SELECT words FROM pos_words;
      DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

      OPEN positive_words_cursor;

      read_loop: LOOP
      FETCH positive_words_cursor INTO positive_word;
      IF done THEN
            LEAVE read_loop;
      END IF;

      IF LOCATE(positive_word, input_text) > 0 THEN
            SET word_count = word_count + 1;
      END IF;
      END LOOP;

      CLOSE positive_words_cursor;

      RETURN word_count;
END//
DELIMITER ;

DROP FUNCTION IF EXISTS `yelp`.`CountNegativeWords`;
```

```
DELIMITER //

CREATE FUNCTION CountNegativeWords(input_text LONGTEXT) RETURNS INT
DETERMINISTIC
BEGIN
      DECLARE word_count INT DEFAULT 0;
      DECLARE current_word VARCHAR(255);
      DECLARE done INT DEFAULT 0;
      DECLARE neg_word_cursor CURSOR FOR
      SELECT words FROM neg_words;
      DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

      SET input_text = REGEXP_REPLACE(input_text, '[[:punct:]]', ' ');
      SET input_text = TRIM(BOTH ' ' FROM input_text);
      SET input_text = CONCAT(' ', input_text, ' ');

      OPEN neg_word_cursor;

      word_loop: LOOP
      FETCH neg_word_cursor INTO current_word;
      IF done THEN
            LEAVE word_loop;
      END IF;

      IF INSTR(input_text, CONCAT(' ', current_word, ' ')) > 0 THEN
            SET word_count = word_count + 1;
      END IF;
      END LOOP;

      CLOSE neg_word_cursor;

      RETURN word_count;
END //

DELIMITER ;


DROP FUNCTION IF EXISTS `yelp`.`wordcount`;

# this function is credited to user Outis on StackExchange
DELIMITER $$
CREATE FUNCTION wordcount(str LONGTEXT)
      RETURNS INT
```

```
      DETERMINISTIC
      SQL SECURITY INVOKER
      NO SQL
  BEGIN
      DECLARE wordCnt, idx, maxIdx INT DEFAULT 0;
      DECLARE currChar, prevChar BOOL DEFAULT 0;
      SET maxIdx=char_length(str);
      SET idx = 1;
      WHILE idx <= maxIdx DO
      SET currChar=SUBSTRING(str, idx, 1) RLIKE '[[:alnum:]]';
      IF NOT prevChar AND currChar THEN
            SET wordCnt=wordCnt+1;
      END IF;
      SET prevChar=currChar;
      SET idx=idx+1;
      END WHILE;
      RETURN wordCnt;
  END
$$
DELIMITER ;
```

```
# 3.3 Find Top 5 Businesses with Highest Average Positive and Negative
Words:
# I created a table with the first 1000 entries in yelp_review.
# Given the massive size of the yelp_review table, the query would take far
too long to finish in today's allotted time
DROP TABLE IF EXISTS yelp_review_small;
CREATE TABLE `yelp_review_small` (
      `review_id` VARCHAR(255) PRIMARY KEY NOT NULL,
      `user_id` VARCHAR(255) NOT NULL,
      `business_id` VARCHAR(255) NOT NULL,
      `stars` INT NULL,
      `date` TEXT NULL,
      `text` LONGTEXT NULL,
      `useful` TEXT NULL,
      `funny` TEXT NULL,
      `cool` TEXT NULL,
      FOREIGN KEY (`user_id`) REFERENCES `yelp_user`(`user_id`),
      FOREIGN KEY (`business_id`) REFERENCES `yelp_business`(`business_id`)
);
INSERT INTO `yelp_review_small`
SELECT * FROM `yelp_review`
```

```
LIMIT 1000;

# Now, test my code using the smaller yelp_review table
SELECT business_id, sentiment, avg_words
FROM (
      (SELECT business_id, 'positive' as sentiment,
AVG(CountPositiveWords(text)) AS avg_words
      FROM yelp_review_small
      GROUP BY business_id
      ORDER BY avg_words DESC
      LIMIT 5)

      UNION ALL

      (SELECT business_id, 'negative' as sentiment,
AVG(CountNegativeWords(text)) AS avg_words
      FROM yelp_review_small
      GROUP BY business_id
      ORDER BY avg_words DESC
      LIMIT 5)
) AS subquery;
```

business_id	sentiment	avg_words
k1c_bC3DK6mKg797vH1T8w	positive	58.0000
ZfbwkAOnMguL_znvv_uJZw	positive	37.0000
NA4PCAGkDl90BmLQnVeS-A	positive	36.0000
_DPJ_PHJnXsuhYhsgBA0Xw	positive	35.0000
NASjUYKkz6l7hQLxMf2lqg	positive	34.0000
PAjWYqHa9xjDUs_sXHFuWg	negative	19.0000
TovMdKATgbYoKi3TEYOEwg	negative	18.0000
O3BrC1MXb3GaRQyQ43v0Qw	negative	17.0000
rPSHxmJNLKNVPK5XUlBpFg	negative	15.0000
ip-0MsQaogm18KQhhnTqTQ	negative	14.0000

```
# 3.4 Calculate Sentiment for Each Review:
SELECT
      review_id,
      `text`,
      (COUNTPOSITIVEWORDS(`text`) - COUNTNEGATIVEWORDS(`text`)) /
WORDCOUNT(`text`) AS sentiment
FROM
      yelp_review;
```

	review_id	text	sentiment
▶	___-Bw8LtQgezPiN9xJWaQ	Don't know how I missed this place after so man...	0.0741
	___05rSAAHBiM7XAbXsW-A	I visited Cantina Laredo for a Sunday Brunch an...	0.0319
	___0XFGhjOU1H8Y3cVYjMA	Be aware. There is an extremely limited menu. ...	-0.0455
	___3SR6DPz0F6gLBxxjuVw	This place is amazing. The strawberry cheesec...	0.2188
	___4_AFJm_fOE-HTgPDxjw	i really liked this place.. tequila bar!! how aweso...	0.0638
	___6FK0UHRSWL3kpUvCaEQ	I hate this location so much the manager susan ...	-0.0612
	___aO4EhZULBsJPvJ4rMhA	Weak. This place is a far cry from the Mill Ave. l...	0.0000

QUESTION 4:

```sql
# Q 4.1: Random row sample
SELECT
      id, `name`
FROM
      big_table
ORDER BY RAND()
LIMIT 5; # pick only five
```

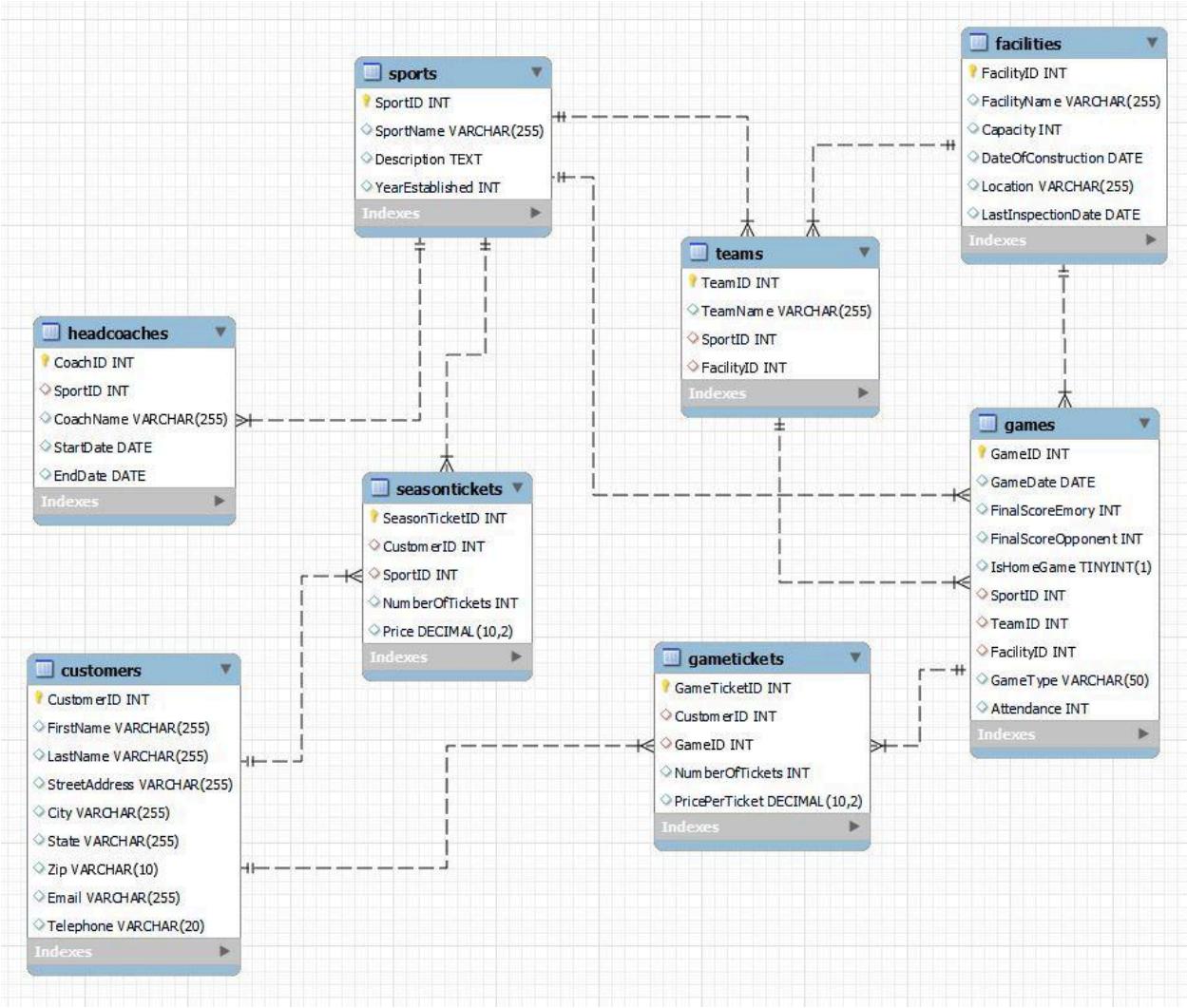
```sql
# Q4.2: Employee Salaries (ETL error)
SELECT
      e.first_name AS first_name,
      e.last_name AS last_name,
      e.salary AS salary
FROM
      employees e
      JOIN
      (SELECT
      first_name, last_name, MAX(id) AS max_id
      FROM
      employees
      GROUP BY first_name , last_name) max_ids
ON e.first_name = max_ids.first_name
      AND e.last_name = max_ids.last_name
      AND e.id = max_ids.max_id;
```

```sql
# 4.3. Monthly Customer Report
SELECT
      DATE_FORMAT(t.created_at, '%Y-%m-01') AS `month`,
      COUNT(DISTINCT u.id) AS num_customers,
      COUNT(DISTINCT t.id) AS num_orders,
      SUM(p.price * t.quantity) AS order_amt
FROM
      transactions t
      JOIN
      users u ON t.user_id = u.id
      JOIN
      products p ON t.product_id = p.id
GROUP BY `month`
ORDER BY `month` ASC;
```

```sql
# 4.4. Closest SAT Score
SELECT
      t1.student AS one_student,
      t2.student AS other_student,
      ABS(t1.score - t2.score) AS score_diff
FROM
      scores AS t1
      JOIN
      scores AS t2 ON t1.id < t2.id
ORDER BY score_diff ASC, t1.id ASC, t2.id ASC
LIMIT 1;
```

```sql
# 4.5. Product Recommendation (co-purchases)
SELECT
      p1.`name` AS P1,
      p2.`name` AS P2,
      COUNT(*) AS count
FROM
      transactions t1
JOIN
      transactions t2 ON t1.user_id = t2.user_id AND t1.product_id <
t2.product_id
JOIN
      products p1 ON t1.product_id = p1.id
JOIN
      products p2 ON t2.product_id = p2.id
GROUP BY
      t1.product_id, t2.product_id
ORDER BY
      COUNT(*) DESC
LIMIT 100;
```

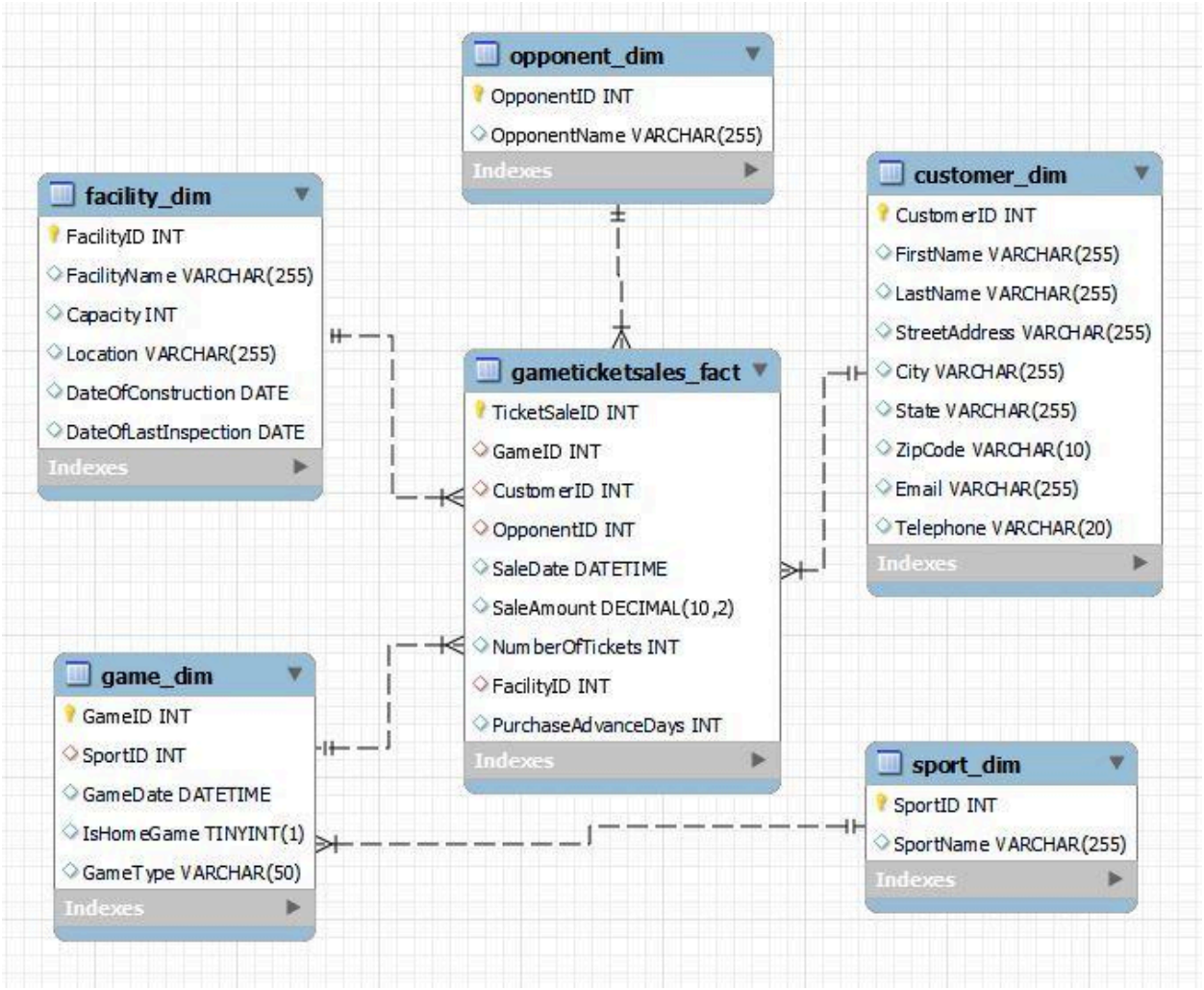
QUESTION 5:
5.1 - 3NF



Assumptions:

- Each head coach can only coach for one sports team
- Each game is played at one facility
- Each team plays in one sport
- Each game is between only two teams
- Each customer can buy multiple tickets for a game or multiple season tickets
- The price for a game ticket can vary for each transaction
- Each game is associated with one specific sport
- Each facility hosts multiple games

5.2:



In my STAR schema (which admittedly is slightly snowflaked, but the professor said this was okay), the gameticketsales_fact table captures individual ticket sale transactions at the granularity of each ticket sold. The fact table is linked to the dimension tables through foreign key relationships, allowing for multi-dimensional analysis based on different attributes like sport, opponent, facility, customer details, and game specifics. This schema enables the Emory University analytics team to easily conduct ad hoc analyses and answer questions, like those specified in this problem, related to ticket sales for different sports games, by accessing the sales data via any one of the IDs present in the fact table.

QUESTION 6:

6.1: NoSQL for Storing User Data

A document-oriented NoSQL database like MongoDB would be suitable for this scenario. In this design, the primary key could be the user ID, ensuring unique identification for each user. The document structure would include fields such as 'user_question' to store the text of user queries, 'user_response' to store the responses (text or images) sent to users, 'timestamp' to record the time of each interaction, and 'clicked_links' to store the URLs of the links clicked by users. Additionally, metadata such as session IDs, device information, and location data can be stored for further analysis. MongoDB's flexible schema would allow for easy adaptation to evolving data requirements and provides efficient querying capabilities, making it an appropriate choice for handling the dynamic and varied nature of user interactions in Bing Chat as the user base grows, ensuring the seamless operation of the integrated solution.

6.2: Deciding on CAP Properties

In the context of storing user data from Bing Chat, the most crucial attributes of the CAP theorem are Consistency and Partition Tolerance. Consistency is vital to ensure that all nodes in the distributed system have the same data at any given time, providing accurate and coherent responses to user queries. Meanwhile, Partition Tolerance is essential for maintaining system functionality, even during network partitions, enabling the database to continue operating seamlessly despite communication failures between nodes. These attributes are paramount for Bing Chat, ensuring both data accuracy and uninterrupted service, ultimately enhancing user satisfaction and reliability.

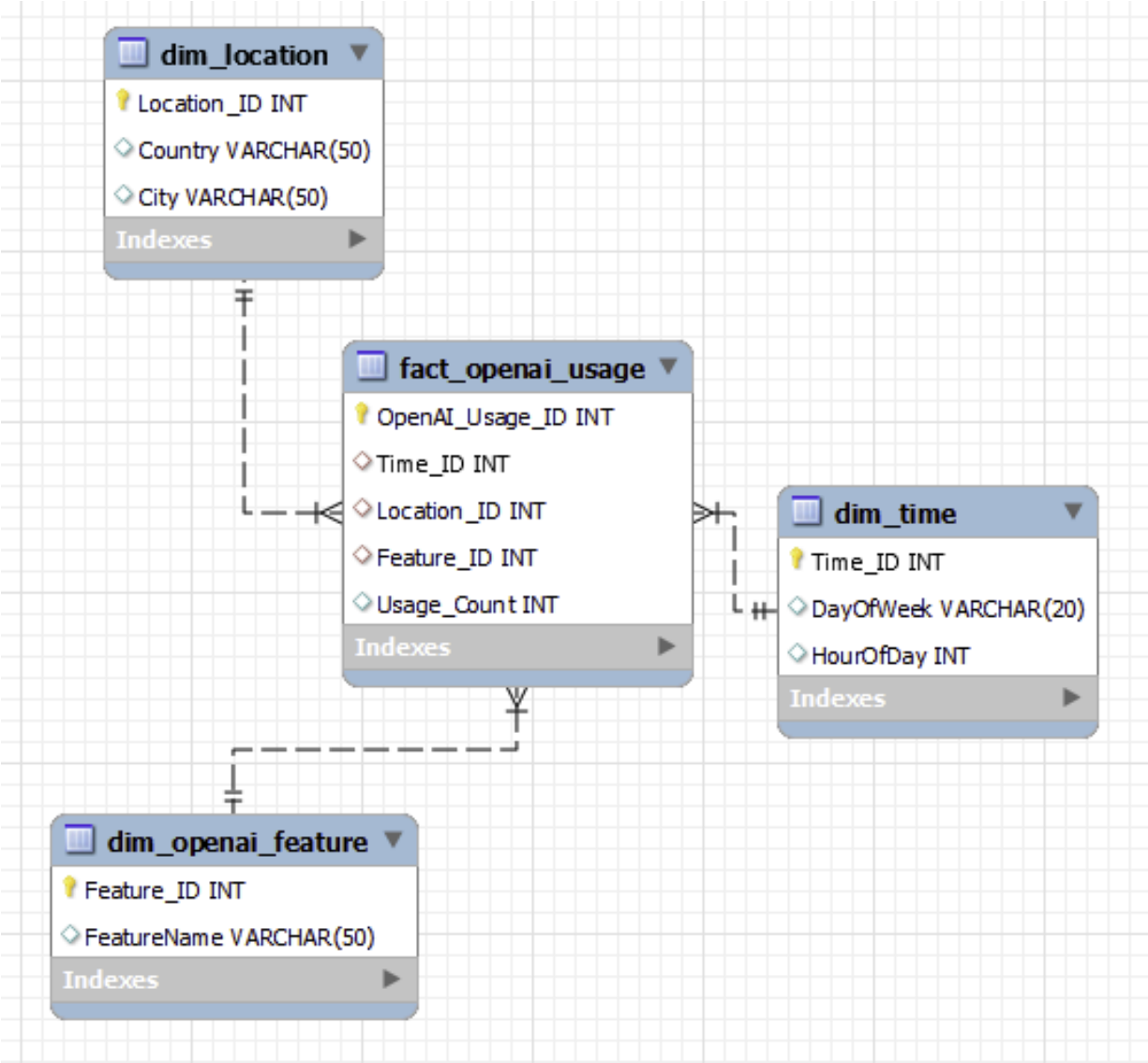
6.3: AI on LinkedIn

Given LinkedIn's current use of the Key-Value NoSQL database Voldemort, integrating OpenAI APIs for new data generated by LinkedIn would require careful consideration. To efficiently handle the diverse and unstructured data generated by OpenAI APIs, a

document-oriented NoSQL database like MongoDB could be a suitable choice. MongoDB allows LinkedIn to store complex data structures, such as user profiles, posts, comments, and AI-generated content, in a flexible and scalable manner. By leveraging MongoDB, LinkedIn can store data generated by OpenAI APIs, including text, images, and other multimedia elements, in a way that accommodates the evolving needs of the platform. MongoDB's ability to handle large volumes of varied data and provide high availability and scalability makes it a relevant choice for LinkedIn's business requirements. If LinkedIn prefers to stick with a Key-Value store, something like HBase could be a viable option, offering robust scalability and fault tolerance while accommodating the influx of data from OpenAI APIs.

6.4: AI Analytics

You can find my STAR schema on the following page. In this schema, the Fact_OpenAI_Usage table serves as the fact table, capturing data about OpenAI usage by referencing three dimension tables: Dim_Time, Dim_Location, and Dim_OpenAI_Feature. The Dim_Time table stores time information, allowing analysis based on both the day of the week and the hour of the day. Dim_Location contains geographical data, specifying the country and city from which OpenAI features are being used. Lastly, Dim_OpenAI_Feature provides a reference for different OpenAI features through their unique Feature_IDs and corresponding feature names. This star schema simplifies complex queries by organizing data into understandable dimensions, which would enable efficient analysis of OpenAI feature usage patterns concerning time, location, and specific features. The schema's structure facilitates streamlined data extraction, transformation, and loading (ETL) processes, ensuring that Microsoft can gain valuable insights into when, where, and how OpenAI features are being utilized.


dim_location
Location_ID INT
Country VARCHAR(50)
City VARCHAR(50)
Indexes
fact_openai_usage
OpenAI_Usage_ID INT
Time_ID INT
Location_ID INT
Feature_ID INT
Usage_Count INT
Indexes
dim_time
Time_ID INT
DayOfWeek VARCHAR(20)
HourOfDay INT
Indexes
dim_openai_feature
Feature_ID INT
FeatureName VARCHAR(50)
Indexes