Prathamesh Mahesh Bhangale

***Database Management Systems***

***Assignment Topic: Implementing a LinkedIn Database***

**ER Diagram:**

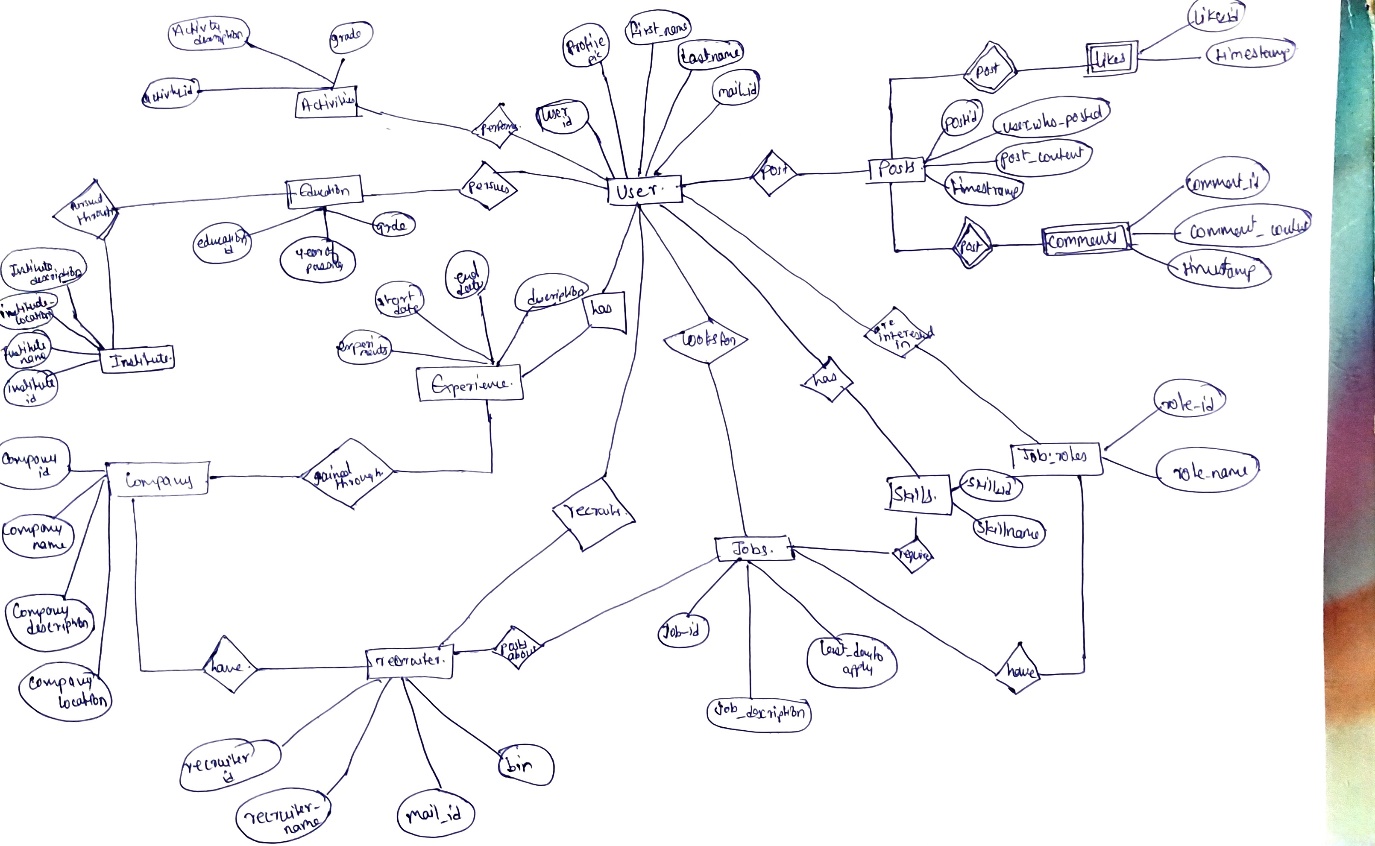
• Create an entity-relationship (ER) diagram to visually represent the

relationships between entities in the LinkedIn database.

• Include entities, attributes, relationships, and cardinalities in the ER

diagram.

• Capture any additional constraints or requirements specific to the

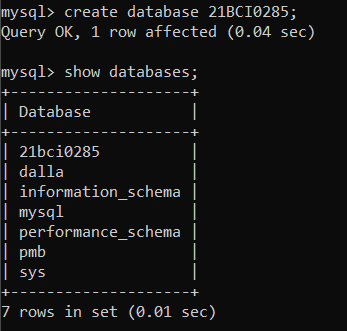
LinkedIn database.

**Database System:**

• Create a new database with your RegNo

Create database 21BCI0285;

Show databases;

******

Design the schema for the LinkedIn database, considering entities

such as users, profiles, connections, job postings, and any other

relevant entities you deem necessary.

***Schema for the LinkedIn Database:***

***Table name:-21BCI0285\_Users***

***Columns:***

|  |  |  |
| --- | --- | --- |
| ***User\_id*** | ***VARCHAR*** | ***Primary Key*** |
| ***First\_name*** | ***VARCHAR*** |  |
| ***Last\_name*** | ***VARCHAR*** |  |
| ***Email*** | ***VARCHAR*** |  |
| ***Password*** | ***VARCHAR*** |  |
| ***Employment\_status*** | ***VARCHAR*** |  |

***Table name: 21BCI0285\_Profiles***

***Columns:***

|  |  |  |
| --- | --- | --- |
| ***profile\_id*** | ***VARCHAR*** | ***Primary Key*** |
| ***User\_id*** | ***VARCHAR*** | ***Foreign key*** |
| ***Headline*** | ***VARCHAR*** |  |
| ***Summary*** | ***VARCHAR*** |  |
| ***experience*** | ***VARCHAR*** |  |

***Table: 21BCI0285\_Connections***

***Columns:***

|  |  |  |
| --- | --- | --- |
| ***connection\_id*** | ***VARCHAR*** | ***Primary Key*** |
| ***User\_id*** | ***VARCHAR*** | ***Foreign key*** |
| ***Connected\_user\_id*** | ***VARCHAR*** | ***Foreign key*** |

***Table: 21BCI0285\_JobPostings***

***Columns:***

|  |  |  |
| --- | --- | --- |
| ***job\_id*** | ***VARCHAR*** | ***Primary Key*** |
| ***User\_id*** | ***VARCHAR*** | ***Foreign key*** |
| ***title*** | ***VARCHAR*** |  |
| ***Location*** | ***VARCHAR*** |  |
| ***Industry*** | ***VARCHAR*** |  |
| ***Salary\_range*** | ***VARCHAR*** |  |

**Create the necessary tables for each entity, ensuring appropriate data types, primary keys, and foreign keys.**

**Implement the necessary integrity constraints, such as unique**

**constraints, not-null constraints, and any other constraints deemed**

**necessary for data consistency.**

*Create table for 21BCI0285\_Users*

Use database 21bci0285;

CREATE TABLE 21BCI0285\_Users (

-> user\_id VARCHAR(50) PRIMARY KEY,

-> first\_name VARCHAR(50),

-> last\_name VARCHAR(50),

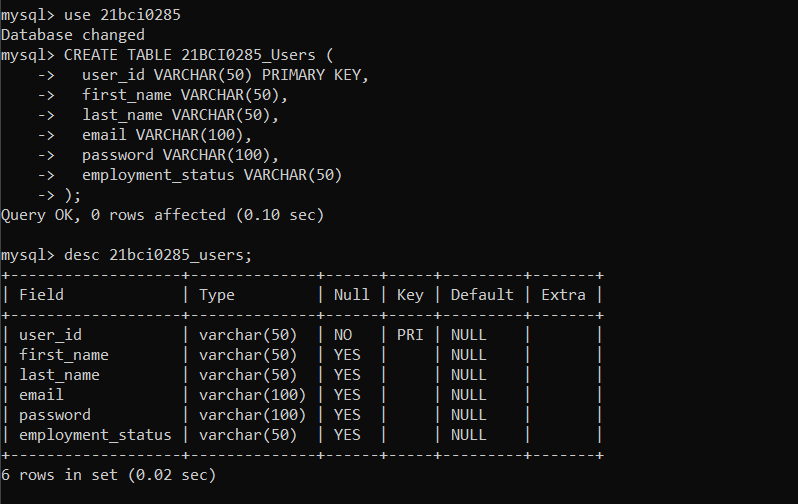
-> email VARCHAR(100),

-> password VARCHAR(100),

-> employment\_status VARCHAR(50)

-> );

desc 21bci0285\_users;



*Create table for 21BCI0285\_Profiles*

CREATE TABLE 21BCI0285\_Profiles (

-> profile\_id VARCHAR(50) PRIMARY KEY,

-> user\_id VARCHAR(50),

-> headline VARCHAR(100),

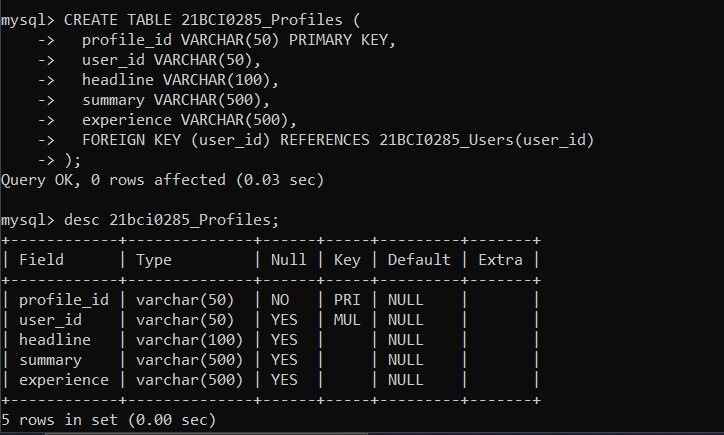
-> summary VARCHAR(500),

-> experience VARCHAR(500),

-> FOREIGN KEY (user\_id) REFERENCES 21BCI0285\_Users(user\_id)

-> );

Desc 21BCI0285\_Profiles;



*Create table for 21BCI0285\_Connections*

CREATE TABLE 21BCI0285\_Connections (

-> connection\_id VARCHAR(50) PRIMARY KEY,

-> user\_id VARCHAR(50),

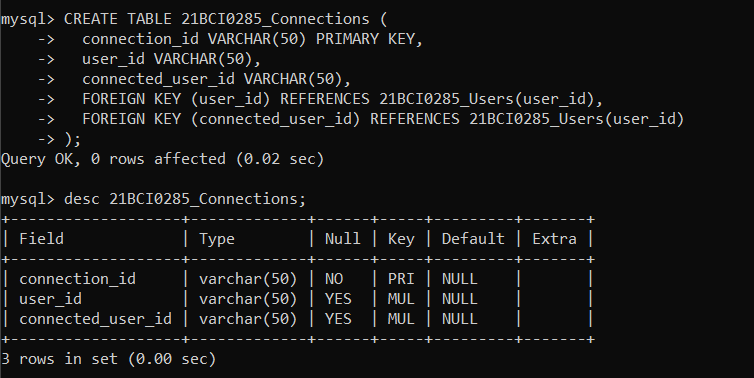
-> connected\_user\_id VARCHAR(50),

-> FOREIGN KEY (user\_id) REFERENCES 21BCI0285\_Users(user\_id),

-> FOREIGN KEY (connected\_user\_id) REFERENCES 21BCI0285\_Users(user\_id)

-> );

Desc 21BCI0285\_Connections;



*Create table for 21BCI0285\_JobPostings*

CREATE TABLE 21BCI0285\_JobPostings (

-> job\_id VARCHAR(50) PRIMARY KEY,

-> user\_id VARCHAR(50),

-> title VARCHAR(100),

-> location VARCHAR(100),

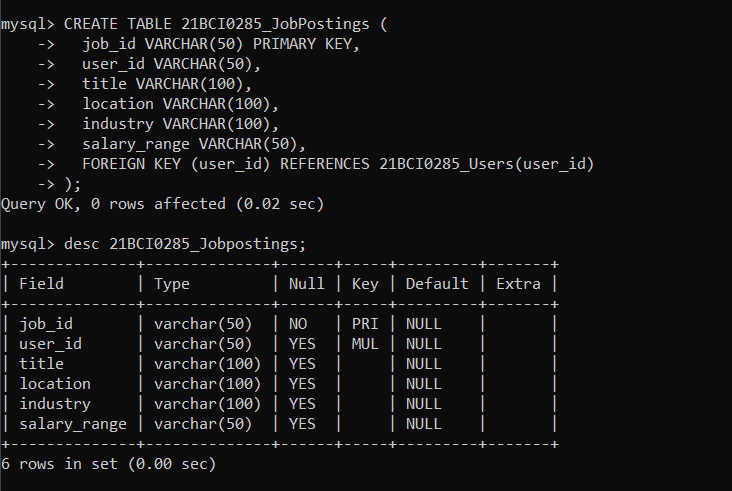
-> industry VARCHAR(100),

-> salary\_range VARCHAR(50),

-> FOREIGN KEY (user\_id) REFERENCES 21BCI0285\_Users(user\_id)

-> );

Desc 21BCI0285\_JOBPostings;



**Populate the tables with sample data to demonstrate the**

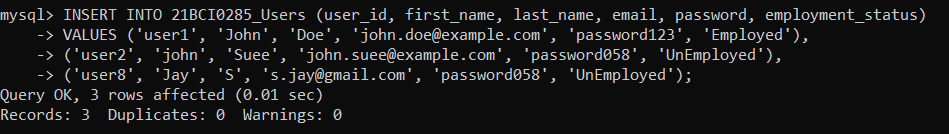
**functionality of the LinkedIn database.**

INSERT INTO 21BCI0285\_Users (user\_id, first\_name, last\_name, email, password, employment\_status)

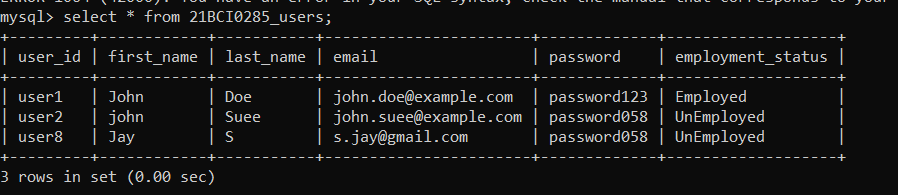
-> VALUES ('user1', 'John', 'Doe', 'john.doe@example.com', 'password123', 'Employed'),

-> ('user2', 'john', 'Suee', 'john.suee@example.com', 'password058', 'UnEmployed'),

-> ('user8', 'Jay', 'S', 's.jay@gmail.com', 'password058', 'UnEmployed');



mysql> select \* from 21BCI0285\_users;



INSERT INTO 21BCI0285\_Profiles (profile\_id, user\_id, headline, summary, experience)

-> VALUES ('profile1', 'user1', 'Software Engineer', 'Experienced software engineer with expertise in web development.', 'Previous job experience');

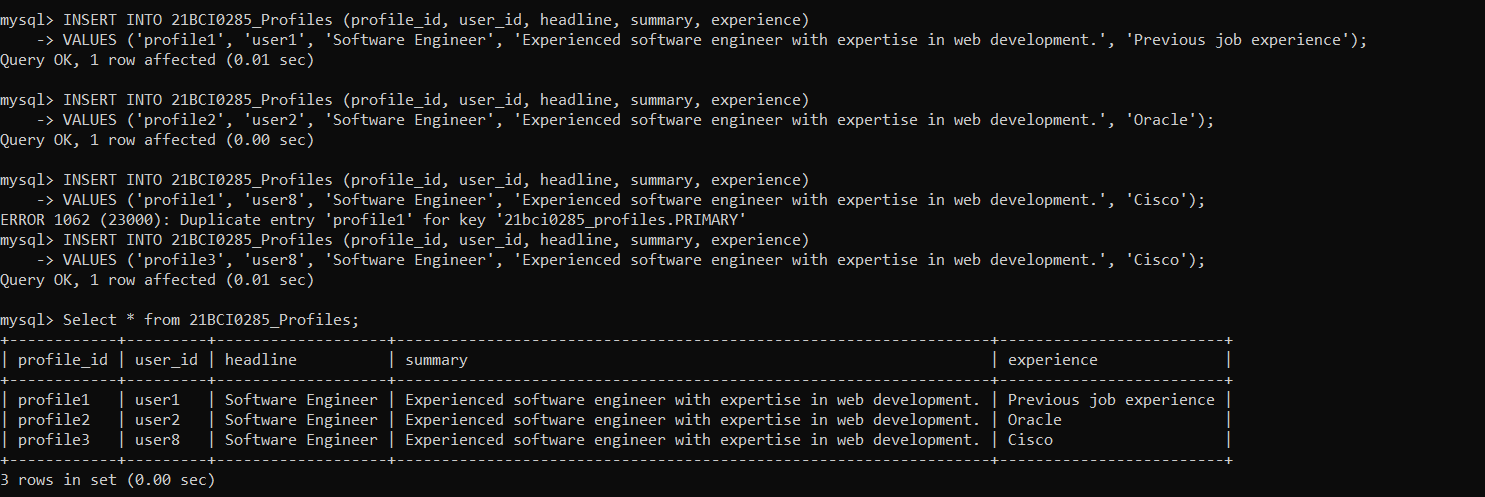
INSERT INTO 21BCI0285\_Profiles (profile\_id, user\_id, headline, summary, experience)

-> VALUES ('profile2', 'user2', 'Software Engineer', 'Experienced software engineer with expertise in web development.', 'Oracle');

INSERT INTO 21BCI0285\_Profiles (profile\_id, user\_id, headline, summary, experience)

-> VALUES ('profile3', 'user8', 'Software Engineer', 'Experienced software engineer with expertise in web development.', 'Cisco');

Select \* from 21BCI0285\_Profiles;



INSERT INTO 21BCI0285\_JobPostings (job\_id, user\_id, title, location, industry, salary\_range)

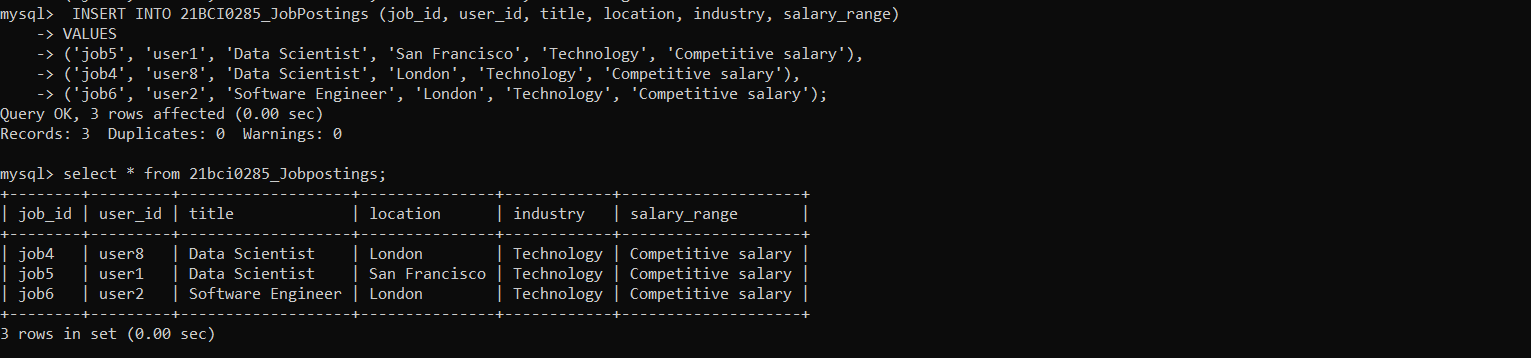
-> VALUES

-> ('job5', 'user1', 'Data Scientist', 'San Francisco', 'Technology', 'Competitive salary'),

-> ('job4', 'user8', 'Data Scientist', 'London', 'Technology', 'Competitive salary'),

-> ('job6', 'user2', 'Software Engineer', 'London', 'Technology', 'Competitive salary');

mysql> select \* from 21bci0285\_Jobpostings;



Write SQL queries to perform the following tasks:

o Retrieve a user's profile information, including their connections and job history.

SELECT Users.\*, Profiles.\*, Connections.\*, Jobpostings.\*

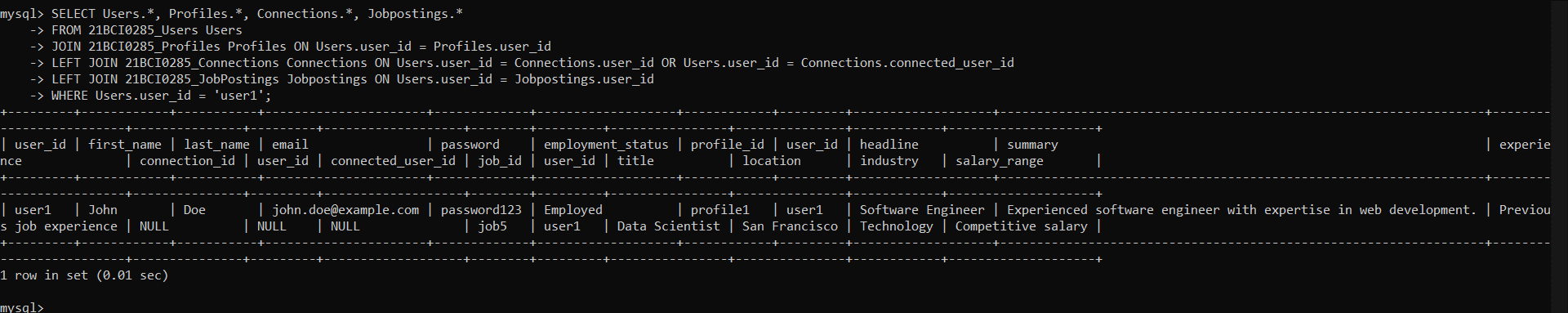
FROM 21BCI0285\_Users Users

JOIN 21BCI0285\_Profiles Profiles ON Users.user\_id = Profiles.user\_id

LEFT JOIN 21BCI0285\_Connections Connections ON Users.user\_id = Connections.user\_id OR Users.user\_id = Connections.connected\_user\_id

LEFT JOIN 21BCI0285\_JobPostings Jobpostings ON Users.user\_id = Jobpostings.user\_id

WHERE Users.user\_id = 'user1';



o Search for job postings based on specific criteria (e.g., job title, location, industry).

SELECT \*

-> FROM 21BCI0285\_JobPostings

-> WHERE title = 'Software Engineer' AND location = 'New York' AND industry = 'Technology';

SELECT \*

-> FROM 21BCI0285\_JobPostings

-> WHERE title = 'Software Engineer' AND location = 'New York' AND industry = 'Technology';

SELECT \*

-> FROM 21BCI0285\_JobPostings

-> WHERE title = 'Software Engineer' AND location = 'London' AND industry = 'Technology';

SELECT \*

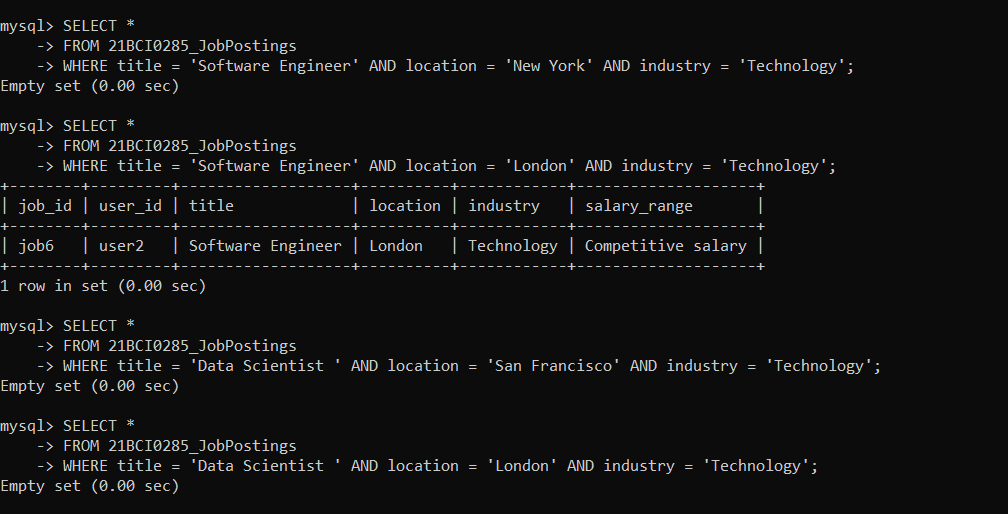
-> FROM 21BCI0285\_JobPostings

-> WHERE title = 'Data Scientist ' AND location = 'San Francisco' AND industry = 'Technology';

SELECT \*

-> FROM 21BCI0285\_JobPostings

-> WHERE title = 'Data Scientist ' AND location = 'London' AND industry = 'Technology';



o Insert new user profiles, job postings, and connection requests into the respective tables.

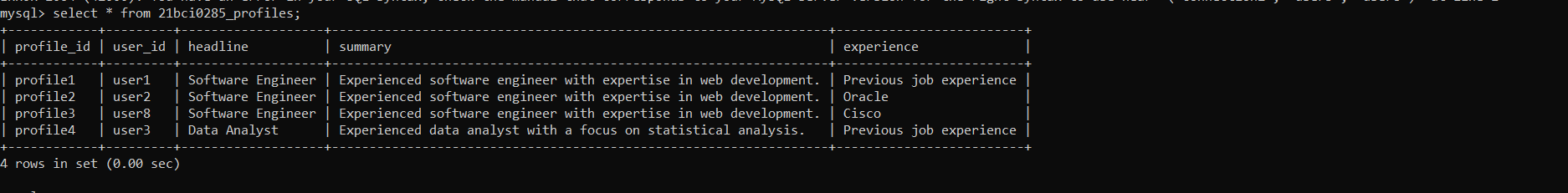
INSERT INTO 21BCI0285\_Users (user\_id, first\_name, last\_name, email, password, employment\_status)

-> VALUES ('user3', 'Mike', 'Johnson', 'mike.johnson@example.com', 'password789', 'Employed');



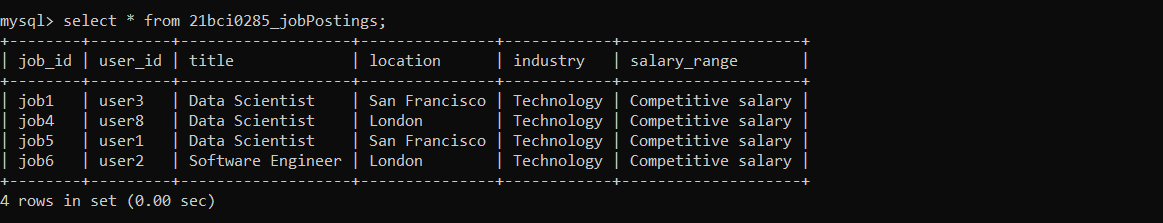
INSERT INTO 21BCI0285\_Profiles (profile\_id, user\_id, headline, summary, experience)

-> VALUES ('profile4', 'user3', 'Data Analyst', 'Experienced data analyst with a focus on statistical analysis.', 'Previous job experience');



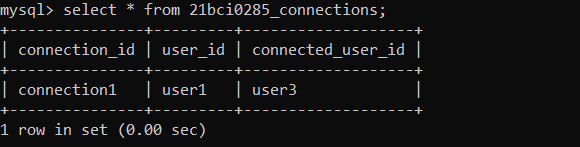
INSERT INTO 21BCI0285\_JobPostings (job\_id, user\_id, title, location, industry, salary\_range)

-> VALUES ('job1', 'user3', 'Data Scientist', 'San Francisco', 'Technology', 'Competitive salary');



INSERT INTO 21BCI0285\_Connections (connection\_id, user\_id, connected\_user\_id)

-> VALUES ('connection1', 'user1', 'user3');

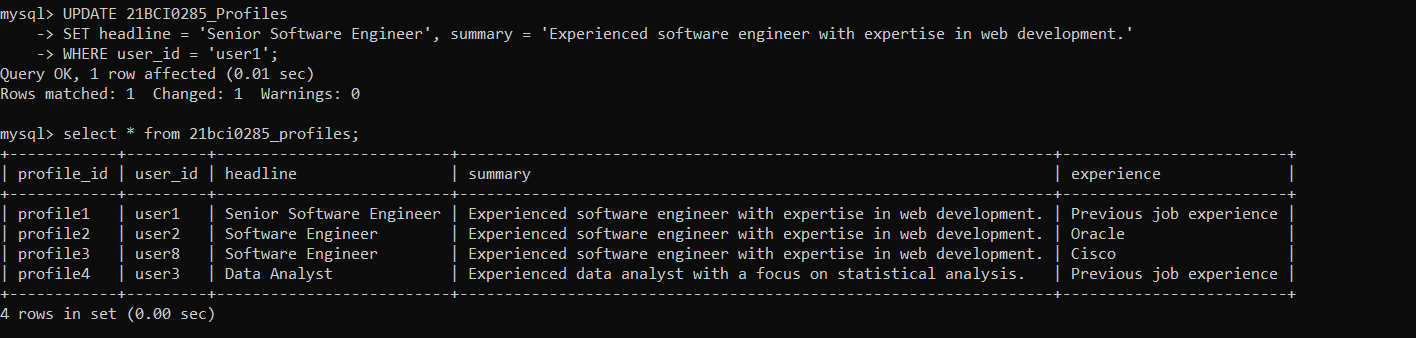


o Update user profile information and job postings.

UPDATE 21BCI0285\_Profiles

SET headline = 'Senior Software Engineer', summary = 'Experienced software engineer with expertise in web development.'

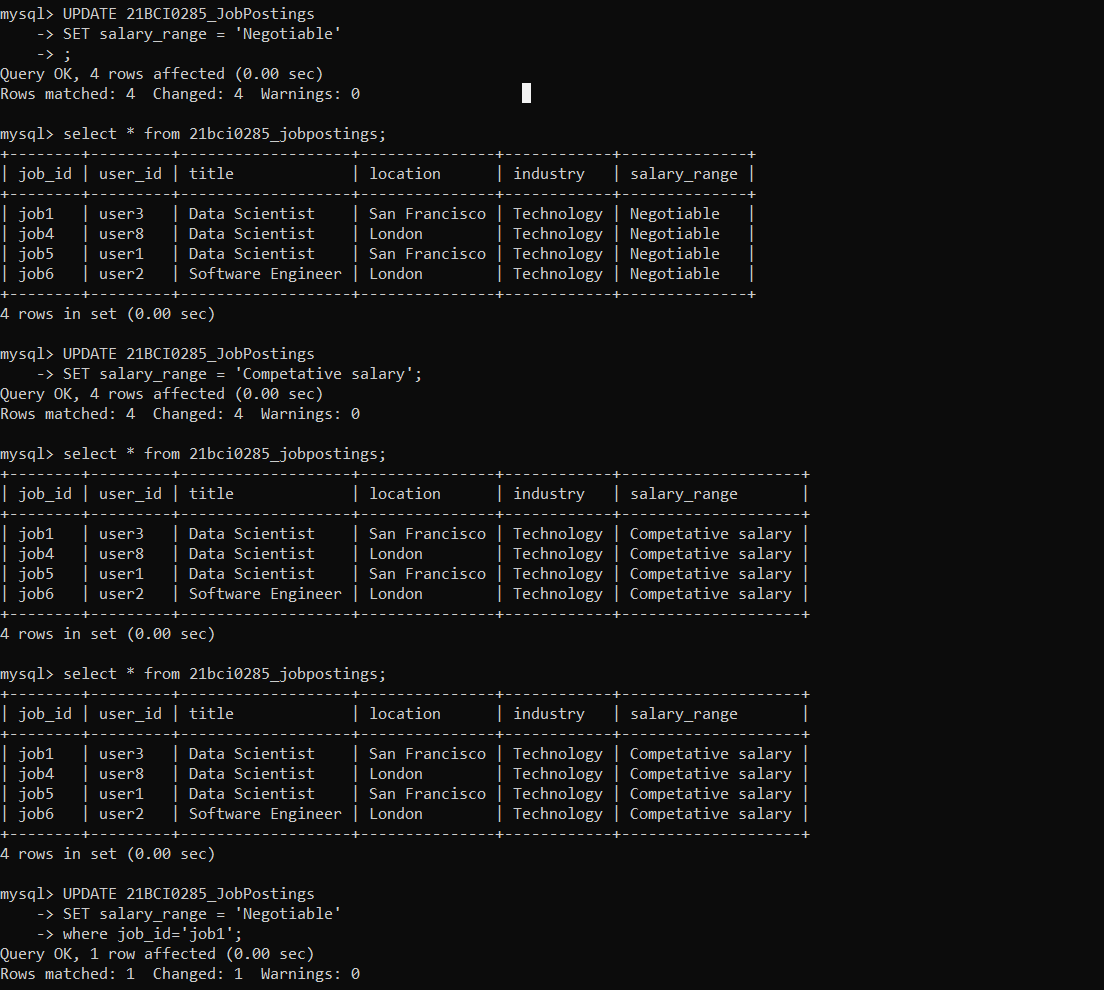
WHERE user\_id = 'user1';



UPDATE 21BCI0285\_JobPostings

SET salary\_range = 'Negotiable'

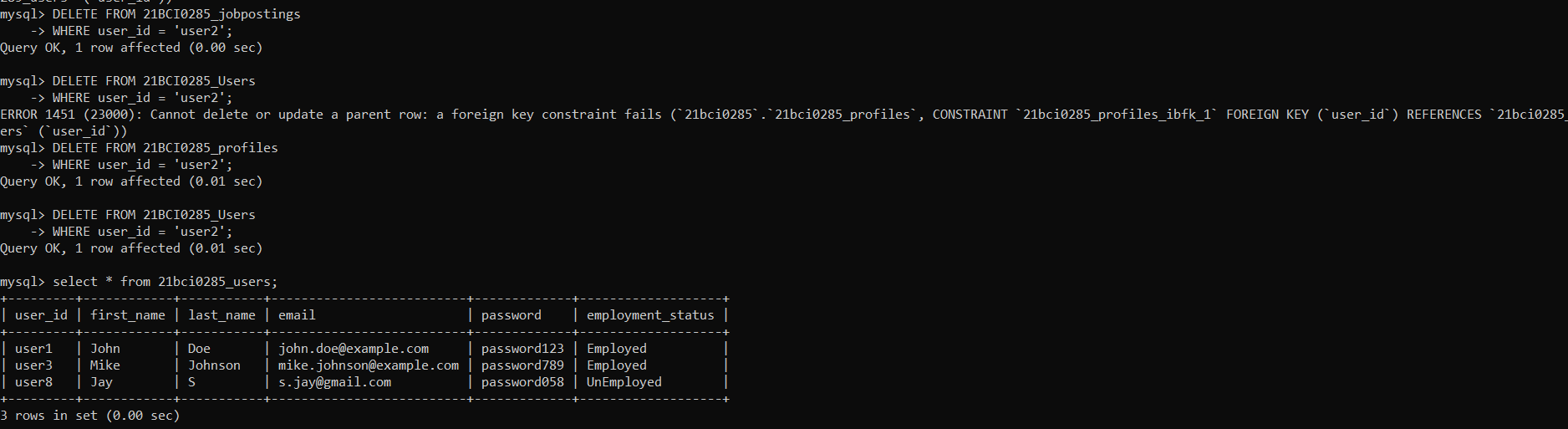
WHERE job\_id = 'job1';



o Delete user profiles and job postings

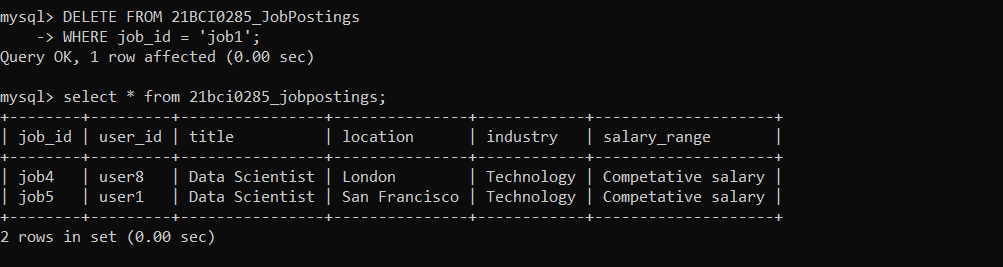
DELETE FROM 21BCI0285\_Users

WHERE user\_id = 'user2';



DELETE FROM 21BCI0285\_JobPostings

WHERE job\_id = 'job1';



Execute and test the SQL queries to ensure they retrieve, modify, and

delete the data accurately.

o Retrieve users who have the most common connections with a

specific user

SELECT User.user\_id, User.first\_name, User.last\_name, COUNT(\*) AS common\_connections

FROM 21BCI0285\_Users User

JOIN 21BCI0285\_Connections C1 ON User.user\_id = C1.connected\_user\_id

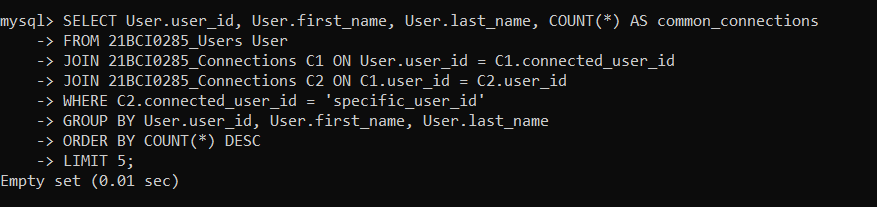
JOIN 21BCI0285\_Connections C2 ON C1.user\_id = C2.user\_id

WHERE C2.connected\_user\_id = 'specific\_user\_id'

GROUP BY User.user\_id, User.first\_name, User.last\_name

ORDER BY COUNT(\*) DESC

LIMIT 5;



o Retrieve users who possess a unique combination of skills

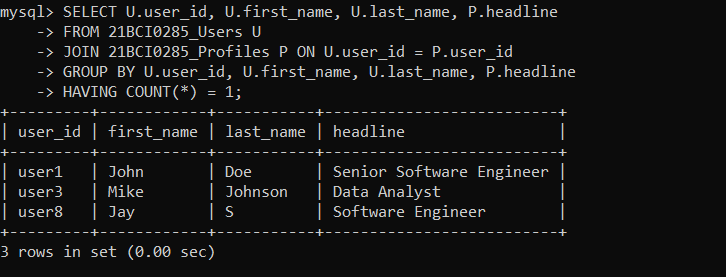
SELECT U.user\_id, U.first\_name, U.last\_name, P.headline

FROM 21BCI0285\_Users U

JOIN 21BCI0285\_Profiles P ON U.user\_id = P.user\_id

GROUP BY U.user\_id, U.first\_name, U.last\_name, P.headline

HAVING COUNT(\*) = 1;



o Calculate the average number of connections per user

SELECT AVG(connection\_count) AS average\_connections

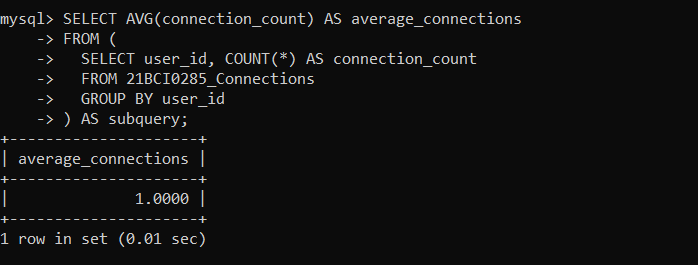
FROM (

SELECT user\_id, COUNT(\*) AS connection\_count

FROM 21BCI0285\_Connections

GROUP BY user\_id

) AS subquery;

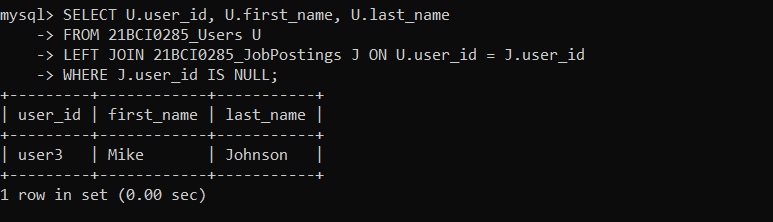


o Retrieve the users who have not posted any job openings

SELECT U.user\_id, U.first\_name, U.last\_name

FROM 21BCI0285\_Users U

LEFT JOIN 21BCI0285\_JobPostings J ON U.user\_id = J.user\_id

WHERE J.user\_id IS NULL;

o Retrieve users who have connections in a specific industry

SELECT DISTINCT U.user\_id, U.first\_name, U.last\_name

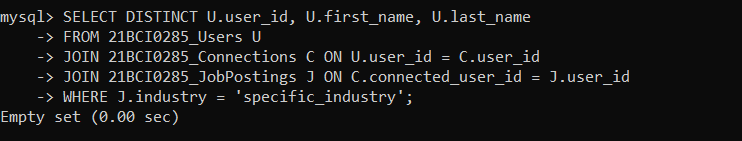
-> FROM 21BCI0285\_Users U

-> JOIN 21BCI0285\_Connections C ON U.user\_id = C.user\_id

-> JOIN 21BCI0285\_JobPostings J ON C.connected\_user\_id = J.user\_id

-> WHERE J.industry = 'specific\_industry';

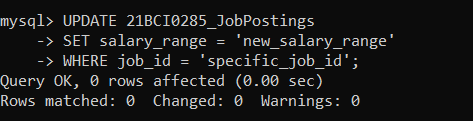
Empty set (0.00 sec)



o Modify a job posting's salary range

UPDATE 21BCI0285\_JobPostings

SET salary\_range = 'new\_salary\_range'

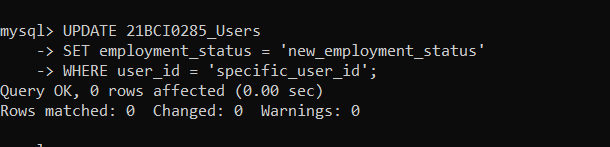
WHERE job\_id = 'specific\_job\_id';

o Modify a user's employment status.

UPDATE 21BCI0285\_Users

SET employment\_status = 'new\_employment\_status'

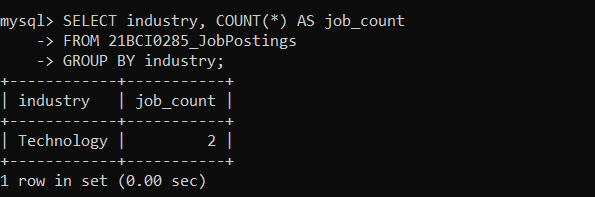
WHERE user\_id = 'specific\_user\_id';



o Retrieve the number of job postings per industry

SELECT industry, COUNT(\*) AS job\_count

FROM 21BCI0285\_JobPostings

GROUP BY industry;

Identify the essential transactions in the LinkedIn database

In the LinkedIn database, some essential transactions can be identified based on the operations performed on the tables. Transactions are units of work that are executed as a single logical operation, ensuring data consistency and integrity. Here are some essential transactions in the LinkedIn database:

1. User Registration Transaction:
   * Inserting a new user into the Users table.
   * Inserting a corresponding profile into the Profiles table.
2. Connection Request Transaction:
   * Inserting a connection request into the Connections table between two users.
3. Job Posting Transaction:
   * Inserting a new job posting into the JobPostings table.
   * Associating the job posting with the user who posted it.
4. Profile Update Transaction:
   * Updating the profile information of a user in the Profiles table.
5. Job Posting Update Transaction:
   * Updating the details of a job posting in the JobPostings table.
6. Connection Acceptance Transaction:
   * Inserting a new connection into the Connections table when a connection request is accepted.
7. User Deletion Transaction:
   * Deleting a user and their associated profile, connections, and job postings from the respective tables.

These are just some examples of essential transactions in the LinkedIn database. The exact transactions may vary depending on the specific requirements and functionality of the LinkedIn platform. It's important to ensure that these transactions are executed atomically and maintain the ACID properties (Atomicity, Consistency, Isolation, Durability) to ensure data consistency and integrity.

Discuss the ACID properties and transaction states.

ACID Properties:

1. Atomicity: Atomicity refers to the "all-or-nothing" property of a transaction. It means that a transaction is treated as a single, indivisible unit of work. Either all the operations within the transaction are successfully completed and committed to the database, or none of them are. If any part of the transaction fails, the entire transaction is rolled back, and the database remains unchanged. This property ensures data consistency and integrity.
2. Consistency: Consistency ensures that a transaction brings the database from one consistent state to another consistent state. It means that a transaction must adhere to any predefined rules or constraints and maintain the integrity of the data. The database should satisfy all integrity constraints before and after the transaction, ensuring that the data is valid and follows the specified rules.
3. Isolation: Isolation ensures that concurrent transactions do not interfere with each other, maintaining data integrity and preventing conflicts. Each transaction executes as if it were the only transaction running on the database, even if multiple transactions are executing concurrently. Isolation guarantees that the intermediate states of a transaction are not visible to other transactions until the transaction is committed, preventing data inconsistencies.
4. Durability: Durability ensures that once a transaction is committed and changes are made to the database, they are permanent and will survive any subsequent system failures, such as power outages or crashes. The committed data is stored persistently, typically through techniques like write-ahead logging or database backups. Durability guarantees that the changes made by a committed transaction are durable and can be recovered in the event of a system failure.

Transaction States:

1. Active: The initial state of a transaction when it starts executing. In this state, the transaction is actively performing operations on the database.
2. Partially Committed: After a transaction has executed all its operations successfully, it enters the partially committed state. At this point, the transaction is ready to commit, but the changes are not yet made permanent.
3. Committed: In the committed state, the transaction's changes are made permanent and durable in the database. Once a transaction is committed, its changes become visible to other transactions.
4. Failed: If an error or exception occurs during the execution of a transaction, it enters the failed state. In this state, the transaction is rolled back, undoing any changes made by the transaction, and bringing the database back to its state before the transaction started.
5. Aborted: When a transaction is intentionally canceled or rolled back, it enters the aborted state. This can happen due to explicit user intervention or because of a system failure.
6. Terminated: The final state of a transaction, either after it has committed successfully or after being rolled back. In this state, the transaction is no longer active or participating in any database operations.