REQUIREMENTS

1) **Execution Steps & Instructions:** Provide a detailed guide on running the submitted code. Ensure the steps include database initialization to prevent duplicate table creation if they already exist.
2) **Normalization & Assumptions:** Explain the normalization techniques applied and any assumptions made during the process.
3) **Challenges & Solutions:** Describe any challenges encountered and how they were resolved

GRADING

1) REQUIRED: Clarity, Completeness, and Justification
2) (5 points) Normalization Process and Assumptions: Clearly stated and justified.
3) (5 points) Referential Integrity Explanation: Demonstrates understanding of constraints and relationships.
4) (5 points) Execution Steps: Data Ingestion, Transformation, and Loading Steps are well-documented
5) (5 points) Challenges & Solutions: Identifies key issues and how they were resolved.
6) (5 points) Overall organization and clarity.
7) DEDUCTIONS: Vague explanations, missing sections, poor organization, or lack of clarity.

Deliverable C: Project Report

Execution Steps & Instructions

1) Data Extraction/Staging: Set up Pentaho and Connect with TablePlus
    a) Download data-integration folder.
    b) Add mysql.jar file to /data-integration/lib/.
    c) Ensure you are using Java 19 in your terminal.
    d) Open up ./spoon.sh to start Pentaho.
    e) Go to File -> Open -> select the .ktr file.
    f) Go to Design -> JSON INPUT and drag into the middle of the dashboard.
    g) Repeat the above with TABLE OUTPUT.
    h) Hold Shift and press to connect JSON input to the table input icons.
    i) Double-click on JSON input POC.
    j) In dir, locate the folder for etl-demo-2025-02-17.
    k) Connect to the src folder than has the company data (company-profile)
    l) Add the following in the input space right below “Regular Expression”: .*\.json
    m) Click “Show filename(s) to ensure they are showing up before proceeding.

n) Click on the “Fields” tab and ensure all twenty-seven attributes are showing up before proceeding.
o) Open TablePlus and select your local DB.
    i) Create a new schema called “gp_01.”
    ii) Add the following query inside that schema to create the “company_raw” table:

```
CREATE TABLE company_raw (
    linkedin_internal_id VARCHAR(255),
    description TEXT,
    website VARCHAR(255),
    industry VARCHAR(255),
    company_size VARCHAR(50),
    company_size_on_linkedin VARCHAR(50),
    hq VARCHAR(255),
    company_type VARCHAR(255),
    founded_year VARCHAR(10),
    specialities TEXT,
    locations TEXT,
    name VARCHAR(255),
    tagline VARCHAR(255),
    universal_name_id VARCHAR(255),
    profile_pic_url VARCHAR(500),
    background_cover_image_url VARCHAR(500),
    search_id VARCHAR(255),
    similar_companies TEXT,
    affiliated_companies TEXT,
    updates TEXT,
    follower_count VARCHAR(50),
    acquisitions TEXT,
    exit_data TEXT,
    extra TEXT,
    funding_data TEXT,
    categories TEXT,
    customer_list TEXT
);
```

    iii) If anything looks amiss at this point, drop the table in the SQL and restart.
p) Go back to the Pentaho dashboard and click on the “TABLE OUTPUT” icon.
q) Under connections, click “EDIT.”
    i) Go to the MySQL connection type and select “Access: Native (JDBC).
    ii) Change to the correct connection DB name (gp_01) and put in your unique password.
    iii) Click “Test” to ensure connection success before proceeding.
r) After ensuring connection success, go to the main dashboard and run the transformation using the play button on the dashboard.
s) Go back to TablePlus and double check that the output has all green checks with no errors.

2) <u>Data Transformation and Loading: SQL Queries</u>
    a) Ensure that the raw data table, aka the staging table, (“company_raw”) was created by running the SQL query (in step 1.o.ii).
        i) NOTE: If any errors are encountered along the way, such as the table already existing, restart by dropping “company_raw”. However, to maintain referential integrity, delete all child tables *first*. There are seven of them that must be dropped before “company_raw.” After dropping the

child tables and “company_raw”, run the creation query again and proceed to the next step.

b) Rename staging table to final table name “company.”
c) Add primary key and additional columns for data normalization.
d) Add columns to store the min and max company size values parsed from the JSON file.
e) Create a “speciality” dimension and “company_specialty” junction.
    i) Stores company specialties in a normalized form.
f) Create a “type” dimension and “company_type” junction.
    i) Normalizes company type information.
g) Create an “industry” dimension and “industry_type” junction.
    i) Normalizes industry information.
h) Create a “locations” dimension and “company_location” junction.
    i) Stores details location information as a one-to-many relationship for each company.
i) Create tables for updates (“company_updates”) and related companies (“affiliated_companies”, “similar_companies”, “similar_companies_junction”).
    i) Stores social media updates and company relationships.
j) Extract data from JSON fields and load into normalized tables.
    i) Parse company size min/max values from JSON.
    ii) Extract and load: speciality data, company type data, industry data, location data, company update data, affiliated companies data, and similar companies data.
k) Clean up by dropping the fifteen redundant columns from the main company table after extraction that have been normalized into separate tables.
l) Remove redundant location fields from related companies tables.

Data Transformation and Cleaning:

- **Elimination of Null Values:**
    - During data cleaning, we observed that the 'location' attribute in the 'Similar_Companies' table and the ‘Affiliated_Companies’ table contained only NULL values. As a result, we removed this attribute to improve data quality and reduce unnecessary storage.
    - We decided to omit the following entities due to a lack of data (NULL values): Acquisitions, Exit_data, Extra, Funding_data, Categories, and Customer_list. If data becomes available, these entities will be incorporated.
- **Attribute Renaming:**
    - To improve clarity and accuracy, we renamed the 'link' attribute to 'linkedin_url' (Similar_Companies table) and the 'line' attribute to 'address_line1'(Locations table). This provides more descriptive attribute names that accurately reflect the data they contain.

- **Table Consolidation:**
    - We identified that the 'Locations' and 'HQ' tables had identical attributes. To eliminate redundancy and simplify the schema, we consolidated these tables into a single 'Locations' table. We utilize the boolean attribute, 'is_hq', to distinguish headquarters locations from other locations, ensuring no data loss.

Normalization & Assumptions:

<u>Assumptions:</u>

Company and Affiliated Companies:

- **Assumption:** A company may or may not have affiliated companies. Affiliated companies are defined as companies directly related to a parent company. An entry in the "Affiliated_Companies" table exists solely because of its relationship to a record in the "Company" table.

Company and Similar Companies (Competitors):

- **Assumption:** Companies may share competitors, and a single company can have multiple competitors.

Company and Locations:

- **Assumption:** Every company must have at least one location. A company can have many locations.

Company and Updates:

- **Assumption:** A company may or may not provide updates. Each update is specific to a single company.

Company and Company Type/Industry/HQ:

- **Assumption:** Each company has one and only one company type, industry, and HQ.

Company and Specialties:

- **Assumption:** A company can have multiple specialties. A specialty can be used by multiple companies.

Normalization:

1. First Normal Form (1NF) Implementation

We transformed the LinkedIn JSON data to 1NF by focusing on the following core principles:

- **Atomic Values:** We ensured that each attribute within every table contains atomic (indivisible) values. This involved examining each field and ensuring that it represented a single, discrete piece of information. (Ex: assigning each attribute in Company table to a single data type value to enforce atomicity)
- **Elimination of Multi-Valued Attributes (MVAs):** The original JSON data contained numerous arrays of dictionaries representing multi-valued attributes (e.g., locations, updates, similar companies). To eliminate these MVAs, we created separate, related tables. Each item within an original array was transformed into a single record in its respective table.
- **Primary Key Establishment:** To uniquely identify each record in every table, we established primary keys (PKs). For example, the "Company" table was assigned a "company_id" as its primary key. This step is essential for establishing relationships between tables and ensuring data integrity.

2. Second Normal Form (2NF) Implementation

- Since our database design employs single-attribute primary keys across all tables, composite keys were not used. Consequently, partial dependencies were not present, and our schema naturally conforms to 2NF.

Handling Many-to-Many Relationships:

- Although not a formal step in the normalization forms (1NF, 2NF, 3NF), we resolved all many-to-many (M:N) relationships to improve data management and query efficiency. We created junction tables to break down each M:N relationship into two one-to-many (1:M) relationships.
    - Ex: the relationship between "Company" and "Similar_Companies" was resolved by introducing a "Similar_Companies_Junction" junction table. This approach eliminates data redundancy and ensures that relationships are clearly defined.

3. Third Normal Form (3NF) Implementation:

- Third Normal Form requires that all non-key attributes be directly dependent on the primary key and not on any other non-key attributes (eliminating transitive dependencies). Upon analysis, we found that our database schema did not contain any transitive dependencies. Therefore, our design inherently satisfies 3NF.
    - Ex: “Affiliated_Companies” table has the following attributes: affiliated_companies_id (PK) , company_id (FK), Name, Linkedin_url, Industry

    - Name, Linkedin_url, industry are directly dependent on affiliated_companies_id.
    - Company_id is a foreign key, and is therefore dependent on the primary key of the company_table, and not a non-key attribute of the Affiliated_companies table.

3. Relationship Implementations

- Many-to-Many relationships were to be resolved though junction tables
- One-to-many relationships were to be implemented via foreign keys
- Proper referential integrity constraints to enforce data consistency
- Insert/delete order is followed (such as parent tables before child tables)

Initially our conceptual model had the following cardinalities/relationships :
Cardinalities (Bidirectional relationships):

- 1 Company has 0 or M specialties, 1 specialty belongs to 1 or M companies (specialty wouldn't exist in specialty table if it didn't belong to one company)
- 1 Company has 1 company type, 1 company type belongs to 1 or M companies
- 1 Company has 1 Industry, 1 Industry has 1 or M companies
- 1 Company has 1 HQ, 1 HQ belongs to 1 Company
- 1 Company has 1 or M Locations, 1 Location belongs to 1 Company
- 1 Company posts 0 or M Updates, 1 Update belongs to 1 Company (each update is linked to one company)
- 1 Company has 0 or M Affiliated_Companies, 1 Affiliated_Company belongs to 1 Company (ex: Affiliated Companies exist solely through their relationship to a Company)
- 1 Company competes with 0 or M Similar_Companies, 1 Similar_Company is a competitor to 1 or M Companies

4. Forward Engineering

- Transform conceptual MVAs into separate tables
- Resolve M:M relationships with junction tables
- Adding appropriate foreign keys
- Ensure all tables meet 3NF requirements

Challenges & Solutions

1) Challenge: we initially tried to use python scripting for the data transformation stage but ran into some issues getting all of our tables to populate in the script.
    a) Solution: scrapped the python script and decided to directly implement all of our queries in TablePlus.

2) Challenge: after modifying “company_size,” we were unsure if we could instead store both values in the range in the same table to retain simplicity and discard unnecessary joins.
    a) Solution: decided to store the two min and max values of “company_size” as two different attributes in the same table to retain 3NF.

3) Challenge: There were complex many-to-many relationships and they cannot be directly implemented in relational databases
    a) Solution: We create junction tables with composite keys as mentioned earlier. Each junction table contains foreign keys to both related entities resolving the many-to-many relationship into two one-to-many relationships

4) Challenge: Maintaining data consistency when inserting or deleting related records across multiple tables
    a) Solution: Implemented proper deletion and insertion order as shown in the lecture slides. Parent records must be created before child records, and child records must be deleted before parent records