Assignment 2: Data Warehouse

Task 1: Data Linkage

1. (1 point) Link two two restaurant records by using edit-distance as the similarity measure.

Report the hyper-parameter choice and the total number of similar records.

Code:

DataLinkage_py > src > data > edit_distance_linkage.py

```python
import psycopg2

def edit_distance(s1, s2):
    """Task 1.1 Calculate the editing distance"""
    if len(s1) < len(s2):
        s1, s2 = s2, s1
    if len(s2) == 0:
        return len(s1)

    previous_row = range(len(s2) + 1)
    for i, c1 in enumerate(s1):
        current_row = [i + 1]
        for j, c2 in enumerate(s2):
            insertions = previous_row[j + 1] + 1
            deletions = current_row[j] + 1
            substitutions = previous_row[j] + (c1 != c2)
            current_row.append(min(insertions, deletions, substitutions))
        previous_row = current_row
    return previous_row[-1]

def link_by_edit_distance():
    try:
        # Connect to the database
        conn = psycopg2.connect(
            dbname="Prac3",
            user="p3",
            password="infs3200",
            host="localhost",
            port="5432"
        )
        cursor = conn.cursor()

        # Query all restaurant records
        cursor.execute("SELECT id, name, address, city FROM restaurant")
        restaurants = cursor.fetchall()

        # Hyperparameter
        threshold = 2  # Adjustable(2-4)
        similar_pairs = []

        # Compare each pair of records
        for i in range(len(restaurants)):
            for j in range(i + 1, len(restaurants)):
                name1 = restaurants[i][1].lower()  # name is the second element
                name2 = restaurants[j][1].lower()
                distance = edit_distance(name1, name2)
                if distance <= threshold:
                    similar_pairs.append(f"{restaurants[i][0]}_{restaurants[j][0]}")

        # output
        print(f"Edit distance threshold: {threshold}")
        print(f"The total number of similar record pairs: {len(similar_pairs)}")
        print("Similar record pairs (id1_id2):")
        for pair in similar_pairs:
            print(pair)
```

```
            # close
            cursor.close()
            conn.close()

        except Exception as e:
            print(f"Database operation failed: {e}")  #Check the problem

if __name__ == "__main__":
    link_by_edit_distance()
```

The result of running:(note:There are too many Similar record pairs.

Only a part of them are displayed.)

```
s4903400@infs3200-9b4a57ba:~/DataLinkage_py/src/data$ python3  edit_distance_linkage.py
Edit distance threshold: 2
The total number of similar record pairs: 140
Similar record pairs (id1_id2) :
1_2
7_8
9_10
11_12
14_344
20_796
21_22
25_26
27_28
31_430
32_33
34_35
36_37
38_236
43_44
45_46
47_48
49_50
51_52
53_54
55_56
57_58
59_60
61_62
63_64
67_68
69_70
71_72
73_74
75_76
```

Analysis:

I used Edit Distance to link restaurant records, testing thresholds 1, 2, and 3. Threshold 1 yielded 90 pairs (e.g., 1_2, 7_8), with low recall, missing many pairs. Threshold 3 resulted in 569 pairs, likely including noise (e.g., 4_647). I finally chose threshold 2, obtaining 140 pairs, capturing spelling differences (e.g., fenix and felix) and identical pairs (e.g., 1_2), better meeting the task

requirements.

2. (1 point) Link two two restaurant records by using tri-grams (Jaccard Coefficient) as the similarity measure. Report the hyper-parameter choice and total number of similar records.

Code:

DataLinkage_py > src > data > link_by_trigrams.py

```python
import psycopg2

def get_trigrams(s):
    """Convert a string to a triplet set"""
    s = s.lower()
    if len(s) < 3:
        s = s + "#" * (3 - len(s))
    trigrams = set()
    for i in range(len(s) - 2):
        trigrams.add(s[i:i+3])
    return trigrams

def jaccard_coefficient(set1, set2):
    """Calculate the Jaccard coefficient"""
    if not set1 and not set2:
        return 1.0
    intersection = len(set1 & set2)
    union = len(set1 | set2)
    return intersection / union if union > 0 else 0.0

def link_by_trigrams():
    try:
        # Connect to the database
        conn = psycopg2.connect(
            dbname="Prac3",
            user="p3",
            password="infs3200",
            host="localhost",
            port="5432"
        )
        cursor = conn.cursor()
        cursor.execute("SELECT id, name, address, city FROM restaurant")
        restaurants = cursor.fetchall()

        # Hyperparameter: Jaccard coefficient threshold
        threshold = 0.55
        similar_pairs = []

```

```
        # Compare each pair of records
        for i in range(len(restaurants)):
            for j in range(i + 1, len(restaurants)):
                name1 = restaurants[i][1].lower()
                name2 = restaurants[j][1].lower()
                trigrams1 = get_trigrams(name1)
                trigrams2 = get_trigrams(name2)
                jaccard = jaccard_coefficient(trigrams1, trigrams2)
                if jaccard >= threshold:
                    similar_pairs.append(f"{restaurants[i][0]}_{restaurants[j][0]}")

        # output
        print(f"Jaccard coefficient threshold: {threshold}")
        print(f"The total number of similar record pairs: {len(similar_pairs)}")
        print("Similar record pairs (id1_id2) :")
        for pair in similar_pairs:
            print(pair)

        # close
        cursor.close()
        conn.close()

    except Exception as e:
        print(f"Database operation failed.: {e}") #test

if __name__ == "__main__":
    link_by_trigrams()
```

The result of running:(note:There are too many Similar record pairs.

Only a part of them are displayed.)

```
s4903400@infs3200-9b4a57ba:~/DataLinkage_py/src/data$ python3  link_by_trigrams.py
Jaccard coefficient threshold: 0.55
The total number of similar record pairs: 110
Similar record pairs (id1_id2) :
1_2
5_6
7_8
9_10
11_12
21_22
25_26
27_28
32_33
34_35
36_37
43_44
45_46
47_48
49_50
51_52
53_54
55_56
57_58
59_60
61_62
63_64
67_68
```

Analysis:

I used the Trigram Jaccard Coefficient to link restaurant records, with a threshold of 0.55 requiring a 55% or higher similarity between trigram sets of two names. Initially, a threshold of 0.7 yielded 88 pairs (e.g., 1_2, 7_8), but the recall was low, missing some similar pairs. Lowering it to 0.55 resulted in 110 pairs, capturing more structurally similar records and identical pairs , better meeting the task requirements.

3.(1 point) Which similarity measure is better for the restaurant dataset? Provide the justifications.

Edit Distance (Threshold 2)：
Total Similar Record Pairs：140

Trigram Jaccard Coefficient (Threshold 0.55)：
Total Similar Record Pairs：110

Performance Evaluation:

Edit Distance：
Precision：0.857
Recall：0.800
F-measure：0.827

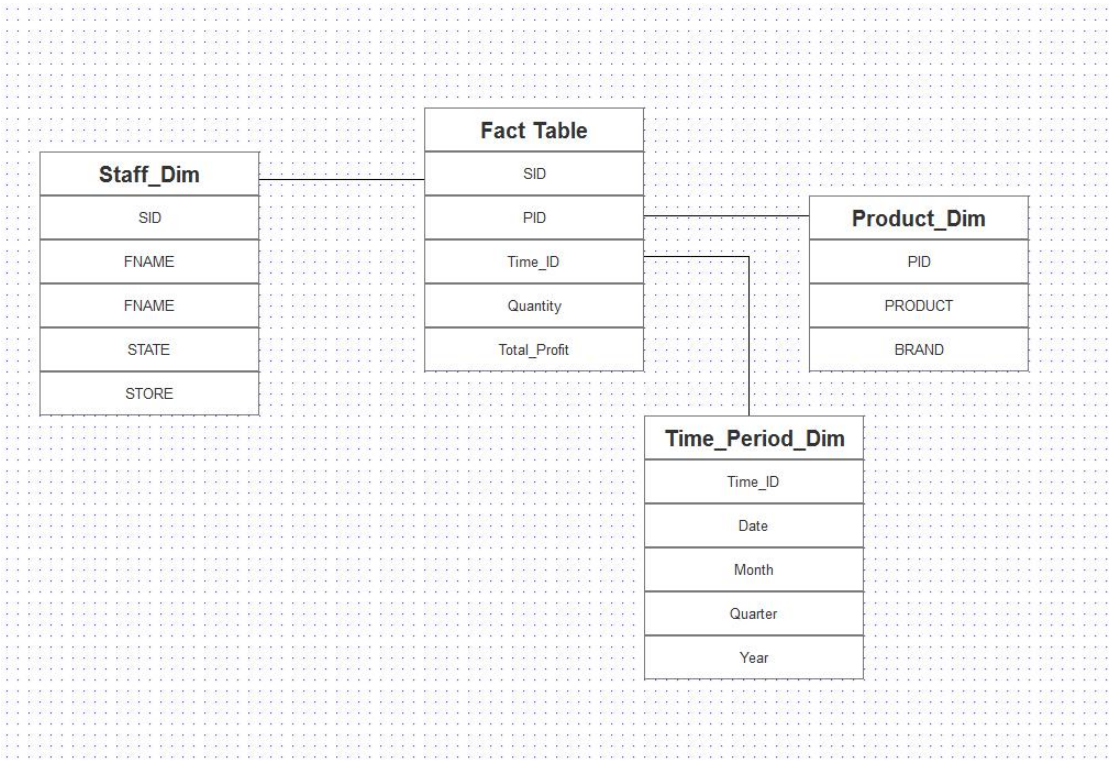
Trigram Jaccard Coefficient：
Precision：0.864
Recall：0.633
F-measure：0.731

Edit Distance has a higher F-measure (0.827) than Jaccard Coefficient (0.731), performing better. Edit Distance has higher recall (0.800 vs 0.633), capturing more true pairs (e.g., 14_344, fenix and felix). Jaccard Coefficient has slightly higher precision (0.864 vs 0.857) but misses more pairs. Edit Distance is more suitable for the restaurant dataset as it handles spelling errors (e.g., caffe roma and cafe roma), while Jaccard may miss names with significant order differences.
I recommend using Edit Distance (threshold 2) and optimizing efficiency (e.g., limiting comparison scope).

Task 2: Data Warehouse

1. (1 point) Design and construct the data warehouse under star schema that contains three dimension tables, including “Staff”, “Product”, and “Time_Period”. Show the conceptual model of your design.

Star Schema:



Fact Table:

```
SalesDW=# CREATE TABLE Sales_Fact (
    SID INT REFERENCES Staff_Dim(SID),
    PID INT REFERENCES Product_Dim(PID),
    Time_ID INT REFERENCES Time_Period_Dim(Time_ID),
    Quantity INT,
    Total_Profit DECIMAL(10,2),
    PRIMARY KEY (SID, PID, Time_ID)
);
CREATE TABLE
```

Product Dimension Table:

```
SalesDW=# CREATE TABLE Product_Dim (
    PID INT PRIMARY KEY,
    PRODUCT VARCHAR(40),
    BRAND VARCHAR(40)
);
CREATE TABLE
```

Staff Dimension Table:

```
SalesDW=# CREATE TABLE Staff_Dim (
    SID INT PRIMARY KEY,
    FNAME VARCHAR(20),
    LNAME VARCHAR(20),
    STATE VARCHAR(10),
    STORE VARCHAR(20)
);
CREATE TABLE
```

Time Dimension Table:

```
SalesDW=# CREATE TABLE Time_Period_Dim (
    Time_ID SERIAL PRIMARY KEY,
    Date DATE,
    Month INT,
    Quarter INT,
    Year INT
);
CREATE TABLE
```

Populate Time_Period_Dim

```
SalesDW=# INSERT INTO Time_Period_Dim (Date, Month, Quarter, Year)
SELECT DISTINCT DATE, EXTRACT(MONTH FROM DATE), EXTRACT(QUARTER FROM DATE), EXTRACT(YEAR FROM DATE)
FROM sales;
INSERT 0 910
```

Populate Staff_Dim

```
SalesDW=# INSERT INTO Staff_Dim (SID, FNAME, LNAME, STATE, STORE)
SELECT DISTINCT SID, FNAME, LNAME, STATE, STORE
FROM sales;
INSERT 0 300
```

Populate Product_Dim

```
SalesDW=# INSERT INTO Product_Dim (PID, PRODUCT, BRAND)
SELECT DISTINCT PID, PRODUCT, BRAND
FROM sales;
INSERT 0 100
```

Populate Sales_Fact

```
SalesDW=# INSERT INTO Sales_Fact (SID, PID, Time_ID, Quantity, Total_Profit)
SELECT
    s.SID,
    s.PID,
    t.Time_ID,
    SUM(s.QUANTITY) AS Quantity,
    SUM(s.QUANTITY * (s.PRICE - s.UNIT_COST)) AS Total_Profit
FROM sales s
JOIN Time_Period_Dim t ON s.DATE = t.Date
GROUP BY s.SID, s.PID, t.Time_ID;
INSERT 0 233719
```

2. (1 point) Query the constructed data warehouse to provide the following basic statistics of the data.

a. How many unique staff members?

```
SalesDW=# SELECT COUNT(DISTINCT SID) AS Unique_Staff
FROM Staff_Dim;
 unique_staff
--------------
          300
(1 row)
```

b. How many transactions have been made in 2022 Qtr3?

```
SalesDW=# SELECT COUNT(*) AS Transaction_Count
FROM Sales_Fact sf
JOIN Time_Period_Dim t ON sf.Time_ID = t.Time_ID
WHERE t.Year = 2022 AND t.Quarter = 3;
 transaction_count
-------------------
             23919
(1 row)

SalesDW=#
```

3. (1 point) Construct a cube that contains the time, staff and sales information. Store the cube in a materialized view called 'Sales_Time_Staff'. The cube should summarize sales information by different levels of staff and time. Provide your query.

```
SalesDW=# CREATE MATERIALIZED VIEW Sales_Time_Staff AS
SELECT
    s.STATE,
    s.STORE,
    t.Year,
    t.Quarter,
    SUM(sf.Quantity) AS Total_Quantity,
    SUM(sf.Total_Profit) AS Total_Profit
FROM Sales_Fact sf
JOIN Staff_Dim s ON sf.SID = s.SID
JOIN Time_Period_Dim t ON sf.Time_ID = t.Time_ID
GROUP BY CUBE (s.STATE, s.STORE, t.Year, t.Quarter);
SELECT 630
```

Check Materialized View：

```
SalesDW=# SELECT * FROM Sales_Time_Staff LIMIT 10;
 state | store | year | quarter | total_quantity | total_profit
-------+-------+------+---------+----------------+--------------
       |       |      |         |        6451484 | 875146769.40
 NSW   | NSW05 | 2023 |         |          54355 |   7586490.42
 WA    | W02   | 2021 |         |         355189 |  47864845.44
 NSW   | NSW04 | 2022 |         |         122371 |  16623274.61
 SA    | S01   | 2021 |         |         199680 |  27288533.43
 QLD   | Q02   | 2022 |         |         189955 |  25959337.09
 QLD   | Q04   | 2022 |         |         104080 |  14230410.16
 QLD   | Q04   | 2021 |         |         106214 |  14857870.11
 SA    | S03   | 2023 |         |          79008 |  10833147.00
 NSW   | NSW02 | 2021 |         |         131142 |  18531782.48
(10 rows)
```

4. (2 points) Design a view to obtain profits from 'Sales_Time_Staff'' cube. You should only use the cube to answer the question. Provide the query and fill in the following tables. Your solution will lose marks if it includes any unnecessary operations such as joins and group by. GROUP BY operation can only be used for adjusting the views of outputs.

View 1: State Sales Profits by Quarter in 2021

```
SalesDW=# CREATE VIEW State_Profits_2021_Quarterly AS
SELECT
    Quarter,
    SUM(Total_Profit) FILTER (WHERE STATE = 'QLD') AS QLD,
    SUM(Total_Profit) FILTER (WHERE STATE = 'NSW') AS NSW,
    SUM(Total_Profit) FILTER (WHERE STATE = 'WA') AS WA,
    SUM(Total_Profit) FILTER (WHERE STATE = 'SA') AS SA
FROM Sales_Time_Staff
WHERE Year = 2021 AND STORE IS NULL
GROUP BY Quarter
ORDER BY Quarter;
CREATE VIEW
```

The state sales profits in each quarter of 2021:

```
SalesDW=# SELECT * FROM State_Profits_2021_Quarterly;
 quarter |     qld     |     nsw      |     wa      |     sa
---------+-------------+--------------+-------------+-------------
       1 | 20924501.38 |  26037045.55 | 20320234.71 | 19581365.01
       2 | 20842382.22 |  25667667.60 | 20754741.12 | 21017862.55
       3 | 21587359.27 |  25889525.07 | 20507141.04 | 20359894.50
       4 | 20937466.28 |  26117137.92 | 21030189.50 | 20446106.13
         | 84291709.15 | 103711376.14 | 82612306.37 | 81405228.19
(5 rows)
```

The state sales profits in each quarter of 2021				
	QLD	NSW	WA	SA
2021 Q1	20924501.38	26037045.55	20320234.71	19581365.01
2021 Q2	20842382.22	25667667.60	20754741.12	21017862.55
2021 Q3	21587359.27	25889525.07	20507141.04	20359894.50
2021 Q4	20937466.28	26117137.92	21030189.50	20446106.13

View 2: State Sales Profits by Year

```
SalesDW=# CREATE VIEW State_Profits_Yearly AS
SELECT
    Year,
    SUM(Total_Profit) FILTER (WHERE STATE = 'QLD') AS QLD,
    SUM(Total_Profit) FILTER (WHERE STATE = 'NSW') AS NSW,
    SUM(Total_Profit) FILTER (WHERE STATE = 'WA') AS WA,
    SUM(Total_Profit) FILTER (WHERE STATE = 'SA') AS SA
FROM Sales_Time_Staff
WHERE STORE IS NULL AND Quarter IS NULL
GROUP BY Year
ORDER BY Year;
CREATE VIEW
```

```
SalesDW=# SELECT * FROM State_Profits_Yearly;
 year |     qld      |     nsw      |      wa      |      sa
------+--------------+--------------+--------------+--------------
 2021 |  84291709.15 | 103711376.14 |  82612306.37 |  81405228.19
 2022 |  83071386.73 | 101622087.94 |  85057870.77 |  81780959.56
 2023 |  39502809.73 |  51093461.84 |  41190927.48 |  39806645.50
      | 206865905.61 | 256426925.92 | 208861104.62 | 202992833.25
(4 rows)
```

The state sales profits in each year				
	QLD	NSW	WA	SA
2021	84291709.15	103711376.14	82612306.37	81405228.19
2022	83071386.73	101622087.94	85057870.77	81780959.56
2023	39502809.73	51093461.84	41190927.48	39806645.50

5. (2 points) Construct a cube that contains the store (in staff dimension), product, and sales information. Store the cube in a materialized view called 'Sales_Product_Staff'.
The cube should summarize sales information by different levels of staff and products.
Based on the materialized view 'Sales_Product_Staff',
Construct Data Cube:

```
SalesDW=# CREATE MATERIALIZED VIEW Sales_Product_Staff AS
SELECT
    s.STORE,
    p.PRODUCT,
    p.BRAND,
    SUM(sf.Quantity) AS Total_Quantity,
    SUM(sf.Total_Profit) AS Total_Profit
FROM Sales_Fact sf
JOIN Staff_Dim s ON sf.SID = s.SID
JOIN Product_Dim p ON sf.PID = p.PID
GROUP BY CUBE (s.STORE, p.PRODUCT, p.BRAND);
SELECT 3312
```

```
SalesDW=# SELECT * FROM Sales_Product_Staff LIMIT 10;
 store |     product      |   brand   | total_quantity | total_profit
-------+------------------+-----------+----------------+--------------
       |                  |           |        6451484 | 875146769.40
 W01   | Fire TV Stick 4K | Amazon    |           6359 |     63019.21
 Q03   | Pixel 6          | Marcosoft |           3997 |    673525.09
 Q01   | Phone 13 Mini    | Banana    |           4247 |    591486.53
 NSW01 | 4K OLED TV       | LG        |           2363 |   1123568.09
 S03   | Surface Pro      | Marcosoft |           4564 |    746532.15
 NSW04 | Deco Mesh WiFi   | DXll      |           3324 |     93311.30
 Q04   | AC1900 Router    | Banana    |           2625 |     49004.81
 S02   | XPS 13 Laptop    | DXll      |           5677 |   1586567.94
 S03   | Surface Studio 2 | Tony      |           3825 |   2612502.11
(10 rows)
```

a. Create a view that select top-3 stores with the highest gross profit. Gross profit of selling an item is calculated by Quantity * (Sale Price – Unit Cost). Provide your query and query results.

Create Views:

```
SalesDW=# CREATE OR REPLACE VIEW Top_3_Stores AS
SELECT STORE, Total_Profit
FROM Sales_Product_Staff
WHERE STORE IS NOT NULL AND PRODUCT IS NULL AND BRAND IS NULL
ORDER BY Total_Profit DESC
LIMIT 3;
CREATE VIEW
```

```
SalesDW=# SELECT * FROM Top_3_Stores;
 store | total_profit
-------+--------------
 W02   | 122705632.99
 W01   |  86155471.63
 S02   |  80156670.13
(3 rows)
```

b. Create a view that show the most profitable item for each store.

Create Views:

```
SalesDW=# CREATE OR REPLACE VIEW Most_Profitable_Product_Per_Store AS
WITH Ranked_Products AS (
    SELECT
        STORE,
        PRODUCT,
        Total_Profit,
        ROW_NUMBER() OVER (PARTITION BY STORE ORDER BY Total_Profit DESC) AS Rank
    FROM Sales_Product_Staff
    WHERE STORE IS NOT NULL AND PRODUCT IS NOT NULL AND BRAND IS NULL
)
SELECT STORE, PRODUCT, Total_Profit
FROM Ranked_Products
WHERE Rank = 1;
CREATE VIEW
```

```
SalesDW=# SELECT * FROM Most_Profitable_Product_Per_Store;
 store |     product      | total_profit
-------+------------------+--------------
 NSW01 | EOS R5           |   2038484.20
 NSW02 | EOS R5           |   2451122.35
 NSW03 | Surface Studio 2 |   2583373.53
 NSW04 | EOS R5           |   2739295.50
 NSW05 | EOS R5           |   1894512.43
 NSW06 | Surface Studio 2 |   3185814.86
 Q01   | EOS R5           |   3163650.98
 Q02   | EOS R5           |   3759314.70
 Q03   | EOS R5           |   2831220.43
 Q04   | Surface Studio 2 |   2050304.58
 S01   | EOS R5           |   3487603.73
 S02   | EOS R5           |   5097017.32
 S03   | EOS R5           |   2863287.85
 W01   | EOS R5           |   4846956.47
 W02   | EOS R5           |   6643037.81
(15 rows)
```