Assignment 2: INFS3200

Alexander White: s4321830

July 29, 2025

1 Task 1: Data Linkage

1.1 Link two restaurant records by using edit-distance as the similarity measure. Report the hyper-parameter choice and the total number of similar records.

The edit distance function was implemented using a nested loop approach, similar to that used in the prac sessions. Edit distance between two restaurant names was calculated using the Levenshtein distance formula, with a threshold of 1. Matching record pairs were identified based on this threshold. The number of matched pairs was then evaluated against the gold-standard matches provided in restaurant_pair.csv, and the precision, recall, and F-measure were calculated to assess the performance of the linkage.

The code for the file is in the appendix.

The output is below:

```
s4321830@infs3200-e772e879:~/ass2/DataLinkage_py$ python3 ass2.py
PostgreSQL database version:
('PostgreSQL 17.4 (Ubuntu 17.4-1.pgdg24.04+2) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0, 64-bit',)
Number of matched pairs: 90
Precision= 0.8444444444444444 , Recall= 0.7169811320754716 , Fmeasure= 0.7755102040816326
```

Figure 1: 1.1 Output

Hyper-parameter: 1 (Edit distance)
Total number of similar records: 90

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

A modified version of the provided Jaccard similarity function from the prac was used to compare restaurant names using tri-grams. A similarity threshold of 0.85 was applied, and any pair with a Jaccard coefficient equal to or greater than this value was considered a match.

The number of matched pairs was then evaluated, with precision, recall, and F-measure calculated against the gold standard. The code for the file is in the appendix.

The output is below:

```
PostgreSQL database version:
('PostgreSQL 17.4 (Ubuntu 17.4-1.pgdg24.04+2) on x86_64-pc-linux-gnu, compiled by gcc (Ubuntu 13.3.0-6ubuntu2~24.04) 13.3.0, 64-bit',)
Number of matched pairs: 84
Precision= 0.9166666666666666 , Recall= 0.7264150943396226 , Fmeasure= 0.8105263157894737
```

Figure 2: 1.2 Output

Hyper-parameter: 0.75 (similarity threshold)
Total number of similar records: 84

1.3 Which similarity measure is better for the restaurant database? Provide the justifications

As seen in the outputs above, the tri-gram (Jaccard coefficient) method produced higher precision, recall, and F-measure compared to the edit distance approach using a threshold of 1. While the edit distance method resulted in 90 matched pairs and the tri-gram approach resulted in 84, the quality of the matches was higher for the tri-gram method.

Although edit distance is stricter and often achieves higher precision in theory, it can fail to identify near-duplicate names with minor spelling differences or variations. In this case, it led to lower performance, likely due to misclassifying true duplicates as non-matches. The tri-gram approach, on the other hand, is more robust to such variations by evaluating character-level token overlap, resulting in better overall linkage quality. For these reasons, the tri-gram similarity measure is more suitable for the restaurant dataset.

2 Task 2: Data Warehouse

2.1 Design and construct the data warehouse under star schema that contain three dimension tables, including "Staff", "Product", and "Time_Period". Show the conceptual model of your design.

To follow a star schema for the data warehouse, three dimension tables were created, along with a fact table. These are:

- Staff
- Product
- Time_Period
- (FACT) Sales

This can be shown in the diagram below:

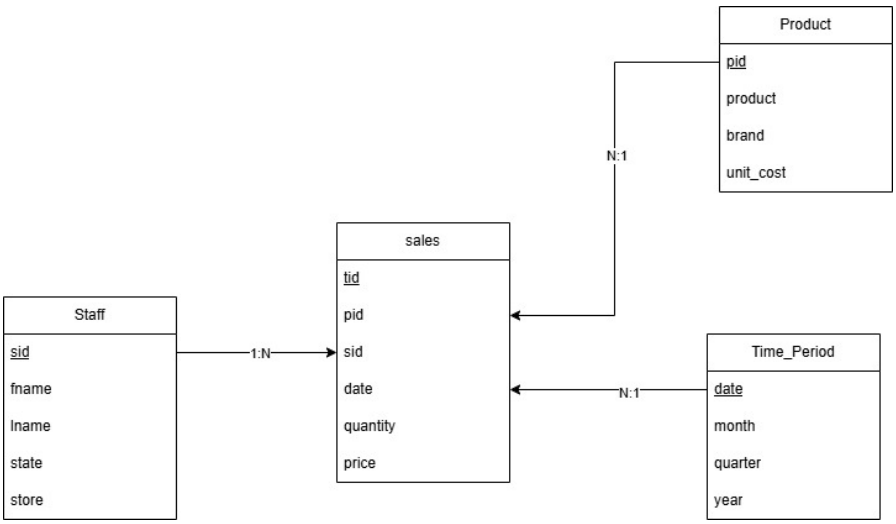

Figure 3: Conceptual Model

This diagram shows how the Staff, Time_Period and Product table are related to the sales table by their primary keys. Below are the screenshots of creating and populating the tables, and the code is in the appendix.

```
s4321830=# CREATE DATABASE ass2;
CREATE DATABASE
s4321830=# \c ass2;
You are now connected to database "ass2" as user "s4321830".
ass2=# CREATE TABLE IF NOT EXISTS "Staff" (
    sid INT PRIMARY KEY,
    fname VARCHAR(20) NOT NULL,
    lname VARCHAR(20) NOT NULL,
    store VARCHAR(20) NOT NULL,
    state VARCHAR(10) NOT NULL
);
CREATE TABLE
ass2=# CREATE TABLE IF NOT EXISTS "Product" (
    pid INT PRIMARY KEY,
    product VARCHAR(40) NOT NULL,
    brand VARCHAR(40) NOT NULL,
    unit_cost DECIMAL(10, 2) NOT NULL
);
CREATE TABLE
ass2=# CREATE TABLE IF NOT EXISTS "Time_Period" (
    date DATE PRIMARY KEY,
    month VARCHAR(20) NOT NULL,
    quarter VARCHAR(20) NOT NULL,
    year INT NOT NULL
);
CREATE TABLE
```

Figure 4: Creating the tables

```
ass2=# CREATE TABLE staging_sales (
    sid INT,
    fname VARCHAR(20),
    lname VARCHAR(20),
    store VARCHAR(50),
    state VARCHAR(30),
    pid INT,
    product VARCHAR(40),
    brand VARCHAR(40),
    unit_cost DECIMAL(10,2),
    quantity INT,
    price DECIMAL(10,2),
    date DATE
);
CREATE TABLE
```

Figure 5: Creating staging table

```
ass2=# \copy staging_sales FROM '/home/s4321830/ass2/DataLinkage_py/Sales.csv' WITH CSV HEADER;
COPY 234680
ass2=# INSERT INTO "Staff" (sid, fname, lname, state, store)
    SELECT DISTINCT sid, fname, lname, state, store
    FROM staging_sales;
INSERT 0 300
ass2=# INSERT INTO "Product" (pid, product, brand, unit_cost)
    SELECT DISTINCT pid, product, brand, unit_cost
    FROM staging_sales;
INSERT 0 100
```

Figure 6: Populating tables with sales data

```
ass2=# INSERT INTO "Time_Period" (date, month, quarter, year)
SELECT DISTINCT
    date,
    TO_CHAR(date, 'Month'),
    'Q' || EXTRACT(QUARTER FROM date),
    EXTRACT(YEAR FROM date)::INT
FROM staging_sales;
INSERT 0 910
```

Figure 7: Populating tables with sales data cont.

```
ass2=# CREATE TABLE sales (
    tid INT PRIMARY KEY,
    pid INT NOT NULL,
    sid INT NOT NULL,
    date DATE NOT NULL,
    quantity INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    CONSTRAINT "sales_fk_prod" FOREIGN KEY (pid) REFERENCES "Product" (pid),
    CONSTRAINT "sales_fk_staff" FOREIGN KEY (sid) REFERENCES "Staff" (sid),
    CONSTRAINT "sales_fk_date" FOREIGN KEY (date) REFERENCES "Time_Period" (date)
);
CREATE TABLE
ass2=# INSERT INTO sales (tid, pid, sid, date, quantity, price)
SELECT tid, pid, sid, date, quantity, price
FROM staging_sales;
INSERT 0 234680
```

Figure 8: Creating fact table

2.2 Query the constructed data warehouse to provide the following basic statistics of the data

2.2.1 How many unique staff members?

```
--How many unique staff members
select count(DISTINCT sid)
from "Staff";
```

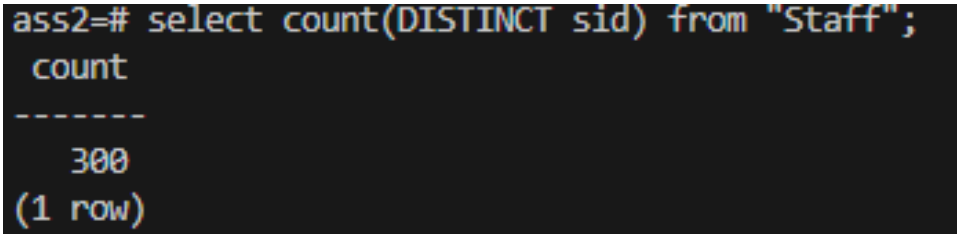

Figure 9: Unique staff query

2.2.2 How many transactions have been made in 2022 Qtr3?

```
-- How many transactions have been made in 2022 Qtr3?
SELECT COUNT(s.tid)
FROM sales s
JOIN "Time_Period" tp ON s.date = tp.date
WHERE tp.year =2022 and tp.quarter = 'Q3';
```

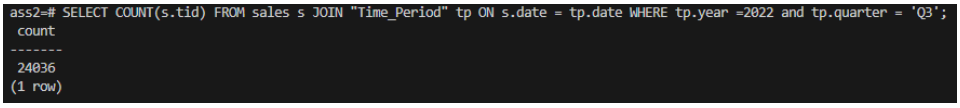

Figure 10: 2022 Q3 Transactions

2.3 Construct a cube that contains the time, staff and sales information.

A materialized view for the data cube was created, using state from the Staff table, and quarter and year from the Time_Period table. This cube allows us to aggregate over those fields, and in this case I added the gross profit margin as the aggregation of choice.

```
CREATE MATERIALIZED VIEW Sales_Time_Staff AS
SELECT
    st.state,
    tp.year,
    tp.quarter,
    SUM(s.quantity*(s.price - p.unit_cost)) AS gross_m
FROM sales s
LEFT JOIN "Time_Period" tp ON s.date = tp.date
LEFT JOIN "Product" p ON s.pid = p.pid
LEFT JOIN "Staff" st ON s.sid = st.sid
GROUP BY CUBE(st.state,tp.quarter, tp.year);
```

```
ass2=# CREATE MATERIALIZED VIEW Sales_Time_Staff AS
SELECT
    st.state,
    tp.year,
    tp.quarter,
    SUM(s.quantity*(s.price - p.unit_cost)) AS gross_m
FROM sales s
LEFT JOIN "Time_Period" tp ON s.date = tp.date
LEFT JOIN "Product" p ON s.pid = p.pid
LEFT JOIN "Staff" st ON s.sid = st.sid
GROUP BY CUBE(st.state,tp.quarter, tp.year);
SELECT 90
```

Figure 11: Creating data cube

2.4 Design a view to obtain profits from 'Sales_Time_Staff' cube.

The following view was made by querying the data cube made from the previous question. This view can be used to get the data needed to populate the tables, but it takes additional wrangling to get it into a format exactly like those tables.

```
CREATE VIEW profits_by_state AS
SELECT
    state,
    year,
    quarter,
    gross_m
FROM Sales_Time_Staff
WHERE state IS NOT NULL
  AND year IS NOT NULL;
```

state	year	quarter	gross_m
SA	2023	Q2	20600706.93
QLD	2021	Q4	20937466.28
NSW	2023	Q1	25124630.30
NSW	2022	Q2	25127473.54
NSW	2022	Q1	25237619.07
SA	2022	Q4	21245225.91
NSW	2022	Q3	25936090.54
NSW	2023	Q2	25968831.54
WA	2022	Q1	21054542.57
SA	2022	Q2	19510444.34
SA	2021	Q4	20446106.13
QLD	2023	Q2	19619967.78
NSW	2021	Q2	25667667.60
WA	2023	Q1	20431691.34
WA	2021	Q2	20754741.12
NSW	2021	Q1	26037045.55
SA	2021	Q1	19581365.01
WA	2021	Q1	20320234.71
QLD	2022	Q3	20742658.37
WA	2021	Q4	21030189.50
WA	2021	Q3	20507141.04
SA	2022	Q3	20619977.05
SA	2021	Q2	21017862.55
QLD	2021	Q2	20842382.22
WA	2022	Q3	21606648.06
WA	2023	Q2	20759236.14
WA	2022	Q2	21175118.89
QLD	2022	Q2	21506540.70
SA	2021	Q3	20359894.50
QLD	2023	Q1	19882841.95
QLD	2022	Q4	20598470.40
QLD	2021	Q3	21587359.27
SA	2022	Q1	20405312.26
QLD	2021	Q1	20924501.38
SA	2023	Q1	19205938.57
NSW	2022	Q4	25320904.79
WA	2022	Q4	21221561.25
NSW	2021	Q4	26117137.92
NSW	2021	Q3	25889525.07
QLD	2022	Q1	20223717.26
SA	2022		81780959.56
NSW	2023		51093461.84
WA	2022		85057870.77
QLD	2022		83071386.73
QLD	2023		39502809.73
NSW	2021		103711376.14
SA	2023		39806645.50
QLD	2021		84291709.15
WA	2023		41190927.48
NSW	2022		101622087.94
WA	2021		82612306.37
SA	2021		81405228.19

(52 rows)

Figure 12: Profits by State results

The queries are in the appendix and the screenshots below show how the view can be filtered to find the data needed to populate the tables:

```
ass2=# SELECT * FROM profits_by_state
WHERE year = 2021 and quarter IS NOT NULL
ORDER BY quarter, state;
 state | year | quarter |   gross_m
-------+------+---------+-------------
 NSW   | 2021 | Q1      | 26037045.55
 QLD   | 2021 | Q1      | 20924501.38
 SA    | 2021 | Q1      | 19581365.01
 WA    | 2021 | Q1      | 20320234.71
 NSW   | 2021 | Q2      | 25667667.60
 QLD   | 2021 | Q2      | 20842382.22
 SA    | 2021 | Q2      | 21017862.55
 WA    | 2021 | Q2      | 20754741.12
 NSW   | 2021 | Q3      | 25889525.07
 QLD   | 2021 | Q3      | 21587359.27
 SA    | 2021 | Q3      | 20359894.50
 WA    | 2021 | Q3      | 20507141.04
 NSW   | 2021 | Q4      | 26117137.92
 QLD   | 2021 | Q4      | 20937466.28
 SA    | 2021 | Q4      | 20446106.13
 WA    | 2021 | Q4      | 21030189.50
(16 rows)

ass2=# SELECT * FROM profits_by_state
WHERE quarter IS NULL;
 state | year | quarter |   gross_m
-------+------+---------+--------------
 SA    | 2022 |         |  81780959.56
 NSW   | 2023 |         |  51093461.84
 WA    | 2022 |         |  85057870.77
 QLD   | 2022 |         |  83071386.73
 QLD   | 2023 |         |  39502809.73
 NSW   | 2021 |         | 103711376.14
 SA    | 2023 |         |  39806645.50
 QLD   | 2021 |         |  84291709.15
 WA    | 2023 |         |  41190927.48
 NSW   | 2022 |         | 101622087.94
 WA    | 2021 |         |  82612306.37
 SA    | 2021 |         |  81405228.19
(12 rows)
```

Figure 13: Profits Queries

This output can then be pivoted to appear like the tables from the question:

```
ass2=# SELECT
    quarter,
    SUM(CASE WHEN state = 'QLD' THEN gross_m ELSE 0 END) AS qld,
    SUM(CASE WHEN state = 'NSW' THEN gross_m ELSE 0 END) AS nsw,
    SUM(CASE WHEN state = 'WA'  THEN gross_m ELSE 0 END) AS wa,
    SUM(CASE WHEN state = 'SA'  THEN gross_m ELSE 0 END) AS sa
FROM profits_by_state
WHERE year = 2021 AND quarter IS NOT NULL
GROUP BY quarter
ORDER BY quarter;
 quarter |     qld     |     nsw     |     wa      |     sa
---------+-------------+-------------+-------------+-------------
 Q1      | 20924501.38 | 26037045.55 | 20320234.71 | 19581365.01
 Q2      | 20842382.22 | 25667667.60 | 20754741.12 | 21017862.55
 Q3      | 21587359.27 | 25889525.07 | 20507141.04 | 20359894.50
 Q4      | 20937466.28 | 26117137.92 | 21030189.50 | 20446106.13
(4 rows)

ass2=# SELECT
    year,
    SUM(CASE WHEN state = 'QLD' THEN gross_m ELSE 0 END) AS qld,
    SUM(CASE WHEN state = 'NSW' THEN gross_m ELSE 0 END) AS nsw,
    SUM(CASE WHEN state = 'WA'  THEN gross_m ELSE 0 END) AS wa,
    SUM(CASE WHEN state = 'SA'  THEN gross_m ELSE 0 END) AS sa
FROM profits_by_state
WHERE quarter IS NULL
GROUP BY year
ORDER BY year;
 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
(3 rows)
```

Figure 14: Pivoted query results

Table 1: Quarterly Profits by State in 2021

Quarter	**QLD**	**NSW**	**WA**	**SA**
Q1	20,924,501.38	26,037,045.55	20,320,234.71	19,581,365.01
Q2	20,842,382.22	25,667,667.60	20,754,741.12	21,017,862.55
Q3	21,587,359.27	25,889,525.07	20,507,141.04	20,359,894.50
Q4	20,937,466.28	26,117,137.92	21,030,189.50	20,446,106.13

Table 2: Yearly Profits by State (2021–2023)

Year	**QLD**	**NSW**	**WA**	**SA**
2021	84,291,709.15	103,711,376.14	82,612,306.37	81,405,228.19
2022	83,071,386.73	101,622,087.94	85,057,870.77	81,780,959.56
2023	39,502,809.73	51,093,461.84	41,190,927.48	39,806,645.50

2.5 Construct a cube that contains the store, product and sales information.

```
CREATE MATERIALIZED VIEW Sales_Product_Staff AS
SELECT st.store, p.product, SUM(s.quantity*(s.price - p.unit_cost)) AS gross_m
FROM sales s
LEFT JOIN "Staff" st ON s.sid = st.sid
LEFT JOIN "Product" p ON s.pid = p.pid
GROUP BY CUBE(st.store, p.product);
```

```
ass2=# CREATE MATERIALIZED VIEW Sales_Product_Staff AS
SELECT st.store, p.product, SUM(s.quantity*(s.price - p.unit_cost)) AS gross_m
FROM sales s
LEFT JOIN "Staff" st ON s.sid = st.sid
LEFT JOIN "Product" p ON s.pid = p.pid
GROUP BY CUBE(st.store, p.product);
SELECT 1600
```

Figure 15: Creation of Product Cube

2.5.1 Create a view that selects top-3 stores with the highest gross profit.

```
CREATE VIEW top_3_stores AS
SELECT store, gross_m
FROM Sales_Product_Staff
WHERE product IS NULL AND store IS NOT NULL
ORDER BY gross_m DESC
LIMIT 3;
```

```
ass2=# CREATE VIEW top_3_stores AS
SELECT store, gross_m
FROM Sales_Product_Staff
WHERE product IS NULL AND store IS NOT NULL
ORDER BY gross_m DESC
LIMIT 3;
CREATE VIEW
ass2=# select * from top_3_stores;
 store |   gross_m
-------+--------------
 W02   | 122705632.99
 W01   |  86155471.63
 S02   |  80156670.13
(3 rows)
```

Figure 16: View and result for top 3 Stores

2.5.2 Create a view that shows the most profitable item for each store.

```
CREATE VIEW top_product_per_store AS
```

```sql
SELECT store, product, gross_m
FROM (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY store ORDER BY gross_m DESC) AS rn
    FROM Sales_Product_Staff
    WHERE store IS NOT NULL AND product IS NOT NULL
) ranked
WHERE rn = 1;
```

```
ass2=# CREATE VIEW top_product_per_store AS
SELECT store, product, gross_m
FROM (
    SELECT *,
           ROW_NUMBER() OVER (PARTITION BY store ORDER BY gross_m DESC) AS rn
    FROM Sales_Product_Staff
    WHERE store IS NOT NULL AND product IS NOT NULL
) ranked
WHERE rn = 1;
CREATE VIEW
ass2=# select * from top_product_per_store;
 store |     product      |  gross_m
-------+------------------+-----------
 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)
```

Figure 17: View and result for top product by store

3 APPENDIX

3.1 Edit Distance Code

```
import sys
sys.path.append('/home/s4321830/Prac3/DataLinkage_py')
import src.psql.DBconnect as db
from src.data.restaurant import restaurant as res
import datetime
import src.data.csv_loader as csv
import src.data.measurement as measure


def edit_distance(str1, str2):
    m = len(str1)
    n = len(str2)

    # Distance matrix
    dp = [[0 for _ in range (n+1)] for _ in range (m+1)]

    # Base case
    for i in range(m+1):
        dp[i][0] = i
    for j in range(n+1):
        dp[0][j] = j

    # Fill matrix
    for i in range(1, m+1):
        for j in range(1, n+1):
            cost = 0 if str1[i-1]== str2[j-1] else 1
            dp[i][j] = min(
                dp[i-1][j]+1,
                dp[i][j-1]+1,
                dp[i-1][j-1] + cost
            )
    return dp[m][n]

# Nested loop edit distance function

def nested_loop_edit_distance(benchmark_path, t):
    con = db.create_connection()
    cur = con.cursor()
    string_query = "SELECT * FROM RESTAURANT"
    cur.execute(string_query)
    restaurants = []
    for rid, name, address, city in cur:
        restaurant = res()
        restaurant.set_id(rid)
        restaurant.set_name(name)
        restaurant.set_address(address)
        restaurant.set_city(city)
        restaurants.append(restaurant)

    cur.close()
    con.close()
```

```
    results = []
    threshold = t
    for i in range(len(restaurants)):
        restaurant1 = restaurants[i]
        id1 = restaurant1.get_id()
        name1 = restaurant1.get_name()
        for j in range(i + 1, len(restaurants)):
            restaurant2 = restaurants[j]
            id2 = restaurant2.get_id()
            name2 = restaurant2.get_name()

            if name1 and name2:
                dist = edit_distance(name1.lower(), name2.lower())
                if dist <= threshold:
                    results.append(f"{id1}_{id2}")

    print("Number of matched pairs:", len(results))
    benchmark = measure.load_benchmark(benchmark_path)
    measure.calc_measure(results, benchmark)

if __name__ == "__main__":
    nested_loop_edit_distance(
        "/home/s4321830/Prac3/DataLinkage_py/data/restaurant_pair.csv", 1
    )
```

4 Nested loop by name jaccard function

```
def nested_loop_by_name_jaccard(benchmark_path):
    threshold = 0.85
    q = 3

    con = db.create_connection()
    cur = con.cursor()
    string_query = "SELECT * FROM RESTAURANT"
    cur.execute(string_query)
    restaurants=[]
    for rid ,name, address, city  in cur:
        restaurant = res()
        restaurant.set_id(rid)
        restaurant.set_name(name)
        restaurant.set_address(address)
        restaurant.set_city(city)
        restaurants.append(restaurant)

    cur.close()
    con.close()

    results = []
    restaurant1 = res()
    restaurant2 = res()
    id1 = 0
    id2 = 0
    name1 = None
```

```
    name2 = None
    for i in range(0, len(restaurants)):
        restaurant1 = restaurants[i]
        id1 = restaurant1.get_id()
        name1 = restaurant1.get_name()
        for j in range(i + 1, len(restaurants)):
            restaurant2 = restaurants[j]
            id2 = restaurant2.get_id()
            name2 = restaurant2.get_name()
            sim = similarity.calc_jaccard(name1, name2, q)
            if sim >= threshold:
                results.append(str(id1) + '_' + str(id2))

    print("Number of matched pairs:", len(results))
    benchmark = measure.load_benchmark(benchmark_path)
    measure.calc_measure(results, benchmark)


nested_loop_by_name_jaccard("/home/s4321830/Prac3/DataLinkage_py/data/restaurant_p
```

5 Creating data warehouse

```
# Task 2 Data Warewhouse
-- Create the database
CREATE DATABASE ass2;

-- Staff dimension table
CREATE TABLE IF NOT EXISTS "Staff" (
    sid INT PRIMARY KEY,
    fname VARCHAR(20) NOT NULL,
    lname VARCHAR(20) NOT NULL,
    store VARCHAR(20) NOT NULL,
    state VARCHAR(10) NOT NULL
);

-- Product dimension table
CREATE TABLE IF NOT EXISTS "Product" (
    pid INT PRIMARY KEY,
    product VARCHAR(40) NOT NULL,
    brand VARCHAR(40) NOT NULL,
    unit_cost DECIMAL(10, 2) NOT NULL
);

-- Time dimension table
CREATE TABLE IF NOT EXISTS "Time_Period" (
    date DATE PRIMARY KEY,
    month VARCHAR(20) NOT NULL,
    quarter VARCHAR(20) NOT NULL,
    year INT NOT NULL
);

-- Create staging table
```

```
CREATE TABLE staging_sales (
    tid INT,
    sid INT,
    fname VARCHAR(20),
    lname VARCHAR(20),
    state VARCHAR(30),
    store VARCHAR(10),
    date DATE,
    pid INT,
    brand VARCHAR(40),
    product VARCHAR(60),
    unit_cost DECIMAL(10,2),
    quantity INT,
    price DECIMAL(10,2)
);

-- Copy data from csv into staging table

\copy staging_sales FROM '/home/s4321830/ass2/DataLinkage_py/Sales.csv' WITH CSV H


-- Insert data into the dimension tables from the staging table

INSERT INTO "Staff" (sid, fname, lname, state, store)
    SELECT DISTINCT sid, fname, lname, state, store
    FROM staging_sales;

INSERT INTO "Product" (pid, product, brand, unit_cost)
    SELECT DISTINCT pid, product, brand, unit_cost
    FROM staging_sales;

INSERT INTO "Time_Period" (date, month, quarter, year)
SELECT DISTINCT
    date,
    TO_CHAR(date, 'Month'),
    'Q' || EXTRACT(QUARTER FROM date),
    EXTRACT(YEAR FROM date)::INT
FROM staging_sales;

-- Create fact table
CREATE TABLE sales (
    tid INT PRIMARY KEY,
    pid INT NOT NULL,
    sid INT NOT NULL,
    date DATE NOT NULL,
    quantity INT NOT NULL,
    price DECIMAL(10,2) NOT NULL,
    CONSTRAINT "sales_fk_prod" FOREIGN KEY (pid) REFERENCES "Product" (pid),
    CONSTRAINT "sales_fk_staff" FOREIGN KEY (sid) REFERENCES "Staff" (sid),
    CONSTRAINT "sales_fk_date" FOREIGN KEY (date) REFERENCES "Time_Period" (date)
);

-- Insert data into sales fact table from staging table
INSERT INTO sales (tid, pid, sid, date, quantity, price)
    SELECT tid, pid, sid, date, quantity, price
```

```
    FROM staging_sales;
```

6 Data Cube Queries

```
--Sales_Time_Staff cube
DROP MATERIALIZED VIEW Sales_Time_Staff;
CREATE MATERIALIZED VIEW Sales_Time_Staff AS
SELECT
    st.state,
    tp.year,
    tp.quarter,
    SUM(s.quantity*(s.price - p.unit_cost)) AS gross_m
FROM sales s
LEFT JOIN "Time_Period" tp ON s.date = tp.date
LEFT JOIN "Product" p ON s.pid = p.pid
LEFT JOIN "Staff" st ON s.sid = st.sid
GROUP BY CUBE(st.state,tp.quarter, tp.year);

-- Profits by state
CREATE VIEW profits_by_state AS
SELECT
    state,
    year,
    quarter,
    gross_m
FROM Sales_Time_Staff
WHERE state IS NOT NULL
  AND year IS NOT NULL;

-- Query for profits in 2021
SELECT * FROM profits_by_state
WHERE year = 2021 and quarter IS NOT NULL
ORDER BY quarter, state;

-- Query for state sales profits in each year
SELECT * FROM profits_by_state
WHERE quarter IS NULL;

-- Pivoting query to get table view for 2021 query:
SELECT
    quarter,
    SUM(CASE WHEN state = 'QLD' THEN gross_m ELSE 0 END) AS qld,
    SUM(CASE WHEN state = 'NSW' THEN gross_m ELSE 0 END) AS nsw,
    SUM(CASE WHEN state = 'WA'  THEN gross_m ELSE 0 END) AS wa,
    SUM(CASE WHEN state = 'SA'  THEN gross_m ELSE 0 END) AS sa
FROM profits_by_state
WHERE year = 2021 AND quarter IS NOT NULL
GROUP BY quarter
ORDER BY quarter;

-- Pivoting query to get table view for yearly aggregation query:
SELECT
    year,
```

```
    SUM(CASE WHEN state = 'QLD' THEN gross_m ELSE 0 END) AS qld,
    SUM(CASE WHEN state = 'NSW' THEN gross_m ELSE 0 END) AS nsw,
    SUM(CASE WHEN state = 'WA'  THEN gross_m ELSE 0 END) AS wa,
    SUM(CASE WHEN state = 'SA'  THEN gross_m ELSE 0 END) AS sa
FROM profits_by_state
WHERE quarter IS NULL
GROUP BY year
ORDER BY year;
```