In [ ]:

Dataset

In [ ]:
```
with open (r"E:\NPLtoSQL\new3.txt") as file:
    table_structure = file.read()

    print(table_structure)
```

```
Table: Customers
Fields:  CustomerID, CustomerName, Email, PhoneNumber
MainField: CustomerName

Table: Products
Fields:  ProductID, ProductName, CustomerID, Price, StockQuantity
MainField:ProductName

Table: Orders
Fields: OrderID, CustomerID, OrderDate, TotalAmount
MainField: OrderID
```

In [ ]:
```
table_mapping = {}

current_table = None

for line in table_structure.split('\n'):
    if line.startswith('Table:'):
        current_table = line.split(':')[1].strip()
        table_mapping[current_table] = {'fields': [], 'main_field': None}

    elif line.startswith('Fields:'):
        fields = [field.strip() for field in line.split(':')[1].split(',')]
        table_mapping[current_table]['fields'] = fields
    elif line.startswith('MainField:'):
        main_field = line.split(':')[1].strip()
        table_mapping[current_table]['main_field'] = main_field
```

In [ ]:
```
print(table_mapping)
```

```
{'Customers': {'fields': ['CustomerID', 'CustomerName', 'Email', 'PhoneNumber'], 'main_field': 'CustomerName'}, 'Products': {'fields': ['ProductID', 'ProductName', 'CustomerID', 'Price', 'StockQuantity'], 'main_field': 'ProductName'},
'Orders': {'fields': ['OrderID', 'CustomerID', 'OrderDate', 'TotalAmount'], 'main_field': 'OrderID'}}
```

In [ ]:

Checking for Common Field

In [ ]:
```
common_field = set.intersection(*(set(table['fields']) for table in table_mapping.values()))

common_field = ', '.join(common_field)
print(common_field)
```

```
CustomerID
```

In [ ]:

User Question

In [ ]:
```
user_question = "what is the details of customer name, product name and total amount"
```

In [ ]:

Tokenization

In [ ]:
```
from nltk.tokenize import word_tokenize

word_tokens = word_tokenize(user_question)
word_tokens
```

Out[ ]:
```
['what',
 'is',
 'the',
 'details',
 'of',
 'customer',
 'name',
 ',',
 'product',
 'name',
 'and',
 'total',
 'amount']
```

In [ ]:

Pos Tagging

In [ ]:
```
# tagging
from nltk import pos_tag

pos_tagging = pos_tag(word_tokens)
pos_tagging
```

Out[ ]:
```
[('what', 'WP'),
 ('is', 'VBZ'),
 ('the', 'DT'),
 ('details', 'NNS'),
 ('of', 'IN'),
 ('customer', 'NN'),
 ('name', 'NN'),
 (',', ','),
 ('product', 'NN'),
 ('name', 'NN'),
 ('and', 'CC'),
 ('total', 'JJ'),
 ('amount', 'NN')]
```

In [ ]:

In [ ]:

Extracting the Entities from question

In [ ]:
```
from nltk import pos_tag, ne_chunk

chunked = ne_chunk(pos_tagging)

named_entities = []
current_entity = ""

for subtree in chunked:
    if isinstance(subtree, tuple):
        pos, word = subtree
        current_entity += " " + word.lower()
    elif current_entity:
        named_entities.append(current_entity.strip())
        current_entity = ""

if current_entity:
    named_entities.append(current_entity.strip())

adjusted_entities = []

for entity in named_entities:
    entity_words = entity.split()
    adjusted_entity = " ".join([word for word in entity_words if any(word.lower() in field.lower() for table in table_mapping.values() for field in table['fields'])])
    if adjusted_entity:
        adjusted_entities.append(adjusted_entity)

adjusted_entities = [entity.replace(" ", "") for entity in adjusted_entities]

noun_phrases = []
current_phrase = ""

for word, pos_tag in pos_tagging:
    if pos_tag in ['NN', 'NNS', 'NNP', 'JJ', 'NNPS']:
        current_phrase += " " + word.lower()
    elif current_phrase:
        noun_phrases.append(current_phrase.strip())
        current_phrase = ""

if current_phrase:
    noun_phrases.append(current_phrase.strip())

combined_entities = set(adjusted_entities + noun_phrases)

combined_entities = {entity.replace(" ", "") for entity in combined_entities}

print("Combined Entities:", combined_entities)

entities = combined_entities
```

```
Combined Entities: {'totalamount', 'customername', 'productname', 'details'}
```

In [ ]:
```
print(entities)
```

```
{'totalamount', 'customername', 'productname', 'details'}
```

Extracting the Table Names

In [ ]:
```
# Table matching logic based on entities
table_matches = {}

for table, data in table_mapping.items():
    table_matches[table] = 0
    matching_fields = []
    for entity in entities:
        entity_words = entity.split()
        for field in data['fields']:
            field_words = field.lower().replace(" ", "")
            if all(word in field_words for word in entity_words):
                table_matches[table] += 1
                matching_fields.append(field)

    if matching_fields:
        print(f"Table: {table}")
        print("Matching Fields:", matching_fields)
```

```
Table: Customers
Matching Fields: ['CustomerName']
Table: Products
Matching Fields: ['ProductName']
Table: Orders
Matching Fields: ['TotalAmount']
```

In [ ]:
```
from fuzzywuzzy import fuzz

relevant_tables = [table for table, count in table_matches.items() if count > 0]

if not relevant_tables:
    similar_tables = []
    for entity in entities:
        for table in table_mapping.keys():
            if fuzz.partial_ratio(entity, table.lower()) >= 80:
                similar_tables.append(table)

    if similar_tables:
        relevant_tables = similar_tables
    else:
        print("No matching table found")

table_names = ', '.join(relevant_tables)
print(table_names)
```

```
Customers, Products, Orders
C:\Users\kunal\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-packages\Python311\site-packages\fuzzywuzzy\fuzz.py:11: UserWarning: Using slow pure-python SequenceMatcher. Install python-Leven
shtein to remove this warning
  warnings.warn('Using slow pure-python SequenceMatcher. Install python-Levenshtein to remove this warning')
```

In [ ]:
```
if ',' in table_names:
    print("its two tables")
else:
    print("its one tables")
```

```
its two tables
```

Extracting the Column Names

In [ ]:
```
from fuzzywuzzy import fuzz

# Column extraction logic based on the nature of the user question
relevant_columns = set()

if len(relevant_tables) == 1:  # Single Table
    most_relevant_table = relevant_tables[0]
    relevant_fields = table_mapping[most_relevant_table]['fields']

    exact_matches = {entity for entity in entities if entity.replace(" ", "") in [field.lower().replace(" ", "") for field in relevant_fields]}

    similar_words = set()
    for entity in entities:
        for field in relevant_fields:
            if fuzz.partial_ratio(entity.replace(" ", "").lower(), field.replace(" ", "").lower()) >= 80:
                similar_words.add((entity, field))

    relevant_columns_set = exact_matches.union({field for _, field in similar_words})
    relevant_columns = list(relevant_columns_set)

    if "all the columns" in user_question or "list all the" in user_question:
        relevant_columns = relevant_fields

    relevant_columns = list(set(col.lower() for col in relevant_columns))

    if relevant_columns:
        print("Single Table Relevant Columns:", relevant_columns)

    else:
        print("No matching columns found")


else:  # Two Tables
    main_fields = [table_mapping[table]['fields'][0] for table in relevant_tables if table_mapping[table]['fields']]

    if not main_fields:
        main_fields = []

    for entity in entities:
        if entity not in main_fields:
            max_ratio = 0
            best_match = None
            for table in relevant_tables:
                for field in table_mapping[table]['fields']:
                    field_words = field.lower().replace(" ", "")
                    ratio = fuzz.token_sort_ratio(entity, field_words)
                    if ratio > max_ratio:
                        max_ratio = ratio
                        best_match = field
            if best_match:
                relevant_columns.add(best_match)

    relevant_columns = [col for col in relevant_columns if col.lower() not in main_fields]

    filtered_relevant_columns = [col for col in relevant_columns if col.lower() in entities]

    if filtered_relevant_columns:
        print("Two Tables Relevant Columns:", filtered_relevant_columns)
    else:
        print("No matching columns found")

    column_names = filtered_relevant_columns
```

```
Two Tables Relevant Columns: ['TotalAmount', 'ProductName', 'CustomerName']
```

In [ ]:
```
result_string = ', '.join(relevant_columns)
relevant_columns = result_string
relevant_columns

if relevant_columns == '':
    relevant_columns = print(table_names)
    relevant_columns
```

Aggreagate Functions, Where Conditions/ Symbol/ Value, and If Table Joins (TableName.FieldName)

In [ ]:
```
if ',' in table_names:

    fully_qualified_columns = {}

    for table in relevant_tables:
        for entity in entities:
            for field in table_mapping[table]['fields']:
                field_words = field.lower().replace(" ", "")
                if fuzz.partial_ratio(entity, field_words) >= 80:
                    fully_qualified_columns[field] = f'{table}.{field}'


    filtered_columns = {col: fq_col for col, fq_col in fully_qualified_columns.items() if col in column_names}

    for field, fully_qualified_column in fully_qualified_columns.items():
        if isinstance(fully_qualified_column, list):

            for common_field_name in fully_qualified_column:
                print(f"{common_field}: {common_field_name}")
        else:
            print(f"{field}: {fully_qualified_column}")

else:

    aggregates_to_aggregates_function = {
    'many': 'COUNT',
    'total number': 'COUNT',
    'average': 'AVG',
    'highest': 'MAX',
    'maximum': 'MAX',
    'most': 'MAX',
    'lowest': 'MIN',
    'all records': None
    }

    aggregates_functions = None

    if user_question.lower() in aggregates_to_aggregates_function:
        aggregates_functions = aggregates_to_aggregates_function[user_question.lower()]

    else:
        for key, value in aggregates_to_aggregates_function.items():
            if key in user_question.lower():
                aggregates_functions = value
                break


    if aggregates_functions:

        columns_list = [col.strip() for col in relevant_columns.split(',')]

        min_distance = float('inf')
        nearest_column = None

        for column in columns_list:
            keyword_position = user_question.lower().find(aggregates_functions)
            column_position = user_question.lower().find(column.lower())

            distance = abs(column_position - keyword_position)

            if distance < min_distance:
                min_distance = distance
                nearest_column = column

        if nearest_column:
            result = f"{aggregates_functions}({nearest_column})"

            other_columns = [col for col in columns_list if col != nearest_column]
            if other_columns:
                result += f", {', '.join(other_columns)}"

            column = result
        else:
            column = f"{aggregates_functions}({relevant_columns})"
    else:
        column = relevant_columns


    #where condition

    def find_all_nouns_positions(pos_tagging):
        return [i for i, (_, pos) in enumerate(pos_tagging) if pos in ['NN', 'NNS', 'NNP', 'NNPS']]

    def find_nearest_noun(adj_position, nouns_positions, tokens):
        valid_nouns = [pos for pos in nouns_positions if tokens[pos][0].lower() != 'number']

        if not valid_nouns:
            return None

        nearest_noun_position = min(valid_nouns, key=lambda x: abs(x - adj_position))
        return tokens[nearest_noun_position][0]

    nouns_positions = find_all_nouns_positions(pos_tagging)

    comparison_words = {
    "equal to": ["is", "equals", "==", "are", "=="],
    "not_equal to": ["is not", "not equal to", "!=", "are not", "<>", "!="],
    "greater than": ["greater than", "is greater than", "exceeds", "more than"],
    "less than": ["less than", "<", "is less than", "under", "fewer than"],
    "greater than or equal to": ["greater than or equal to", "=", "is at least", "is not less than"],
    "less than or equal to": ["less than or equal to", "<=", "is at most", "is not greater than"],
    "starts with": ["starts with", "begins with", "begins", "starts", "starts by"],
    "with": ["with"],
    "ends with": ["ends with", "concludes with", "finishes with", "ends by"],
    "contains": ["contains", "includes", "has", "contains any of", "includes any of"],
    "is like": ["is like", "resembles", "is similar to", "looks like", "matches"]
    }


    nearest_noun = None

    # Look for comparison words in the user's question
    for comparison, words in comparison_words.items():
        for word in words:
            if word.lower() in user_question.lower():
                adj_position = user_question.lower().index(word.lower())
                nearest_noun = find_nearest_noun(adj_position, nouns_positions, pos_tagging)
                break

    if nearest_noun:
        where_column = nearest_noun.lower() if nearest_noun.lower() in relevant_columns else None
    else:
        where_column = None

    if where_column and where_column not in relevant_columns:
        where_column = None

    # exracting the value, dates

    # where condition syntax

    where_cond = {
        "equal to": "=",
        "not equal to": "!=",
        "greater than": ">",
        "less than": "<",
        "greater than or equal to": ">=",
    }

    # exracting the value, dates ,statuses

    number_where_condition = []

    for word, pos_tag in pos_tagging:
        if pos_tag in ['CD']:
            value = word
            number_where_condition.append(value)
        elif pos_tag in ['NNS', 'NN', 'JJ', 'VBG', 'IN', 'VBN']:
            value = word
            number_where_condition.append(value)


    for condition, symbol in where_cond.items():
        if f"{condition.lower()} " in f" {user_question.lower()} ":
            print(symbol)
            break
    else:
        # Check for "is" condition if no other conditions are found
        if " is " in f" {user_question.lower()} ":
            symbol = "="
        else:
            print("No matching condition found in the user question")

    print("user_question:", user_question)
    print("Table Name:", most_relevant_table)
    print("Column Names:", column)
    print("Where Condition column:",where_column)
    print("Symbol: ",symbol)
    print("Value:",value)
```

```
CustomerID: Orders.CustomerID
CustomerName: Customers.CustomerName
Email: Customers.Email
ProductName: Products.ProductName
TotalAmount: Orders.TotalAmount
```

In [ ]:

Generating Query

In [ ]:
```
if ',' in table_names:

    # Generate SQL query
    sql_query = "SELECT\n"

    for field, fully_qualified_column in filtered_columns.items():
        sql_query += f"    {fully_qualified_column},\n"

    sql_query = sql_query.rstrip(",\n")

    sql_query += "\nFROM\n"

    # Generate JOIN clauses
    selected_tables = list(set([col.split('.')[0] for col in filtered_columns.values()]))
    first_table = selected_tables[0]

    sql_query += f"    {first_table}\n"

    for table in selected_tables[1:]:
        sql_query += f"JOIN\n    {table} ON {table}.{common_field} = {first_table}.{common_field}\n"

    print("Question:", user_question)
    print("--------------------------------")
    print(sql_query)

else:

    #SQL query
    if aggregates_functions:
        sql_query = f"SELECT {column} FROM {most_relevant_table};"
    else:
        sql_query = f"SELECT {column} FROM {most_relevant_table}"
        if where_column and symbol and value:
            sql_query += f" WHERE {where_column} {symbol} {value};"

    print("Question:", user_question)
    print("--------------------------------")
    print(sql_query)
```

```
Question: what is the details of customer name, product name and total amount
--------------------------------
SELECT
    Customers.CustomerName,
    Products.ProductName,
    Orders.TotalAmount
FROM
    Customers
JOIN
    Products ON Products.CustomerID = Customers.CustomerID
JOIN
    Orders ON Orders.CustomerID = Customers.CustomerID
```