AI-Based Natural Language to SQL Query Converter

11th september.2024

Step 1: Prototype Selection

Abstract

The growing trend towards data-driven decision-making in businesses and academic institutions has resulted in an increased need for tools that allow users to query databases efficiently. However, for many individuals and organizations, learning SQL or other querying languages is not feasible due to time constraints or lack of technical expertise. This project proposes the development of a web-based AI service that allows users to input questions in natural language (e.g., English), automatically converts those questions into SQL queries, and retrieves the results from the database. The answers are then presented to the users in natural language.

This solution aims to remove the barrier of learning complex database languages and empower non-technical users to query their databases effectively. It addresses common scenarios such as business analysts wanting quick insights from sales data, marketers seeking customer segmentation insights, or educators looking to analyze student data without needing SQL expertise.

- **Problem Statement**: Today's businesses, from small startups to large enterprises, heavily rely on databases for decision-making processes. However, most professionals in these industries are non-technical and do not possess the knowledge or experience to use SQL or other database querying languages. Our AI service will provide a seamless interface where users can ask questions in English and get responses from the database without needing to know any SQL.

This reduces the training time required for employees, decreases dependency on technical personnel, and improves decision-making speed. It also offers scalability since the same solution can be applied across various industries, including education, healthcare, finance, and retail, where database access is critical.

- **Market Need Assessment**: The global demand for database querying solutions is projected to grow significantly as data continues to proliferate. Non-technical users, such as marketing managers, business owners, and healthcare administrators, require insights from databases but lack the SQL skills to access this data efficiently. This product will cater to these users, enabling them to extract valuable information from databases in a user-friendly way.

In India alone, businesses across sectors such as e-commerce, finance, healthcare, and education are increasingly relying on data analytics. However, the shortage of SQL-trained personnel presents a significant hurdle for companies that wish to leverage their data. A service that bridges this gap will allow them to capitalize on their data assets more effectively.

By eliminating the need to learn SQL or hire additional technical staff, businesses can increase efficiency, reduce costs, and make more informed decisions in real-time.

Prototype Selection Criteria

- **Feasibility**: This product is highly feasible due to the availability of advanced natural language processing (NLP) models such as Google Gemini, OpenAI's GPT models, and other language models that can accurately translate human language into SQL. These models have matured sufficiently in recent years, making it possible to develop and deploy this product within **2-3 years**. Furthermore, integrating the product with common database management systems such as MySQL, PostgreSQL, or SQLite is relatively straightforward using Python and existing libraries.

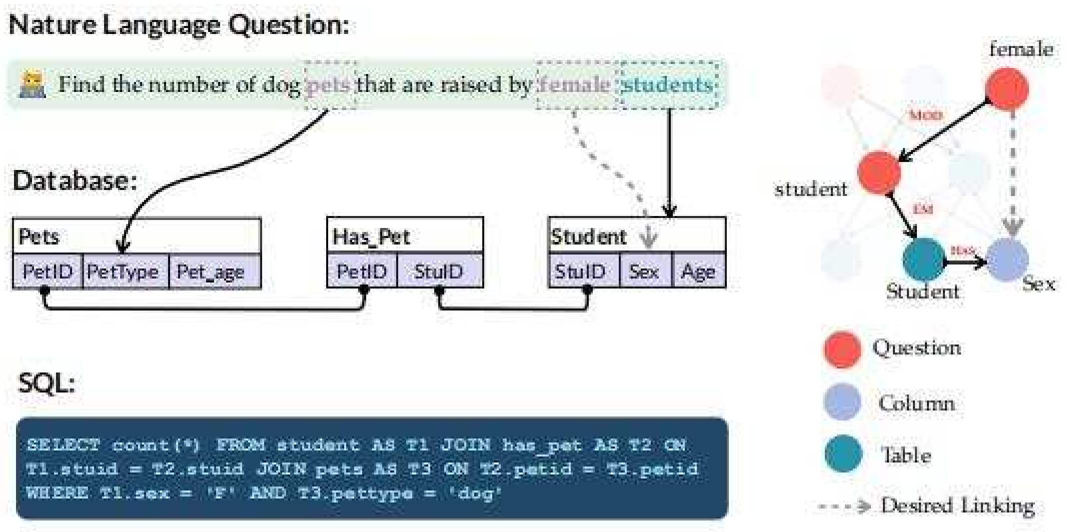
We anticipate that the project will progress smoothly through prototyping, development, and deployment phases due to the ready availability of data storage and querying technologies, cloud infrastructure, and APIs for language model integration. Additionally, the product can be hosted on scalable cloud platforms like AWS or Google Cloud, ensuring it can meet demand as it grows.

- **Viability**: This product is expected to remain viable over the next **20-30 years** due to the increasing reliance on data in business operations and decision-making processes. As more industries adopt AI and machine learning technologies, the demand for tools that simplify data querying and analysis will continue to grow. Additionally, as AI models improve in their understanding of natural language, the service will become even more accurate and capable of handling more complex queries.

Data is a critical asset in the digital age, and as data volumes grow, so will the need for tools that make data more accessible. This product's continuous learning and adaptability will allow it to evolve with technological advancements, ensuring it remains relevant in the long term. By continuously updating the model to support more languages and complex queries, we can ensure the service stays competitive.

- **Monetization**: This product is **directly monetizable** through various channels, with a primary focus on subscription-based services. Businesses and individuals could subscribe to the platform based on their query needs. Different pricing tiers could be implemented depending on the number of queries allowed per month, access to advanced query functionalities (e.g., complex SQL joins, nested queries), and premium features such as predictive analytics.

Additional monetization strategies include offering white-label solutions for large enterprises that want to integrate this functionality into their own platforms. A usage-based pricing model could also be introduced for companies with varying levels of data access requirements, ensuring that even small businesses can afford to use the service. Monetization can also come from partnerships with educational institutions, allowing students to use the tool for learning database queries.



Find the number of dog pets that are raised by female students

```
SELECT count(*) FROM student AS T1 JOIN has_pet AS T2 ON
T1.stuid = T2.stuid JOIN pets AS T3 ON T2.petid = T3.petid
WHERE T1.sex = 'F' AND T3.pettype = 'dog'
```

- **Objectives**:

  - Develop a scalable web application.

  - Integrate the Graphix-T5 model for better parsing accuracy.

  - Support multiple SQL databases.

  - Provide a user-friendly interface, ensuring data security, customization, and comprehensive support.

- **Customer Interviews**:

  - Data analysts, business professionals, and non-technical users stress the need for high accuracy, real-time performance, and ease of use.

- **Key Requirements**:

  - High precision, scalability, security, customization, and a user-friendly interface that supports various databases.

---

3. Concept Generation

- **Brainstorming**:

  - The team generated ideas focusing on a modular architecture, interactive interfaces, and real-time query validation using Graphix-T5.

- **Concept Selection**:

  - Chosen concepts include a web-based interface with a Graphix-T5-powered backend, scalable cloud architecture, and secure access.

- **Final Concept**:

  - The final concept integrates a web interface for user queries, Graphix-T5 for SQL conversion, and database connectors for query execution with real-time feedback.

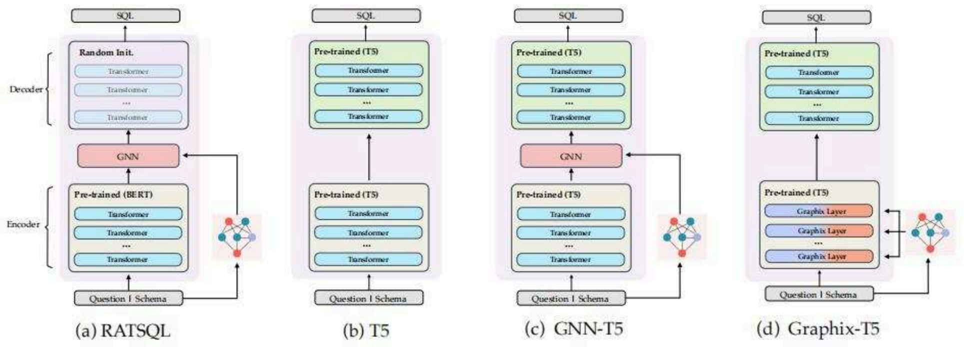
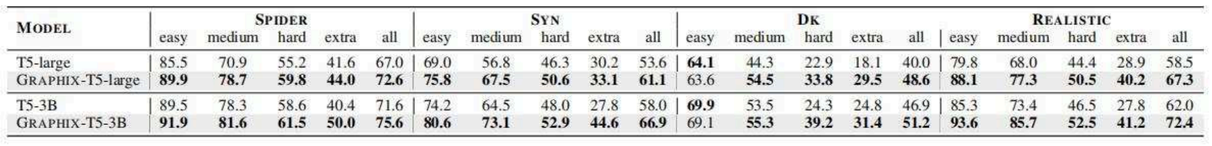


Figure 2: Graphical illustration of existing methods **(a) RATSQL** [pre-trained BERT-encoder → graph-based module → randomly initialized decoder]. **(b) T5** [pre-trained T5-encoder → pre-trained T5-decoder] and the proposed variant **(c) GNN-T5** [pre-trained T5-encoder → graph-based module → pre-trained T5-decoder] **(d) GRAPHIX-T5** [semi-pre-trained graphix-module → pre-trained T5-decoder].



Model	Spider					Syn					Dk					Realistic				
	easy	medium	hard	extra	all	easy	medium	hard	extra	all	easy	medium	hard	extra	all	easy	medium	hard	extra	all
T5-large	85.5	70.9	55.2	41.6	67.0	69.0	56.8	46.3	30.2	53.6	**64.1**	44.3	22.9	18.1	40.0	79.8	68.0	44.4	28.9	58.5
Graphix-T5-large	**89.9**	**78.7**	**59.8**	**44.0**	**72.6**	**75.8**	**67.5**	**50.6**	**33.1**	**61.1**	63.6	**54.5**	**33.8**	**29.5**	**48.6**	**88.1**	**77.3**	**50.5**	**40.2**	**67.3**
T5-3B	89.5	78.3	58.6	40.4	71.6	74.2	64.5	48.0	27.8	58.0	**69.9**	53.5	24.3	24.8	46.9	85.3	73.4	46.5	27.8	62.0
Graphix-T5-3B	**91.9**	**81.6**	**61.5**	**50.0**	**75.6**	**80.6**	**73.1**	**52.9**	**44.6**	**66.9**	69.1	**55.3**	**39.2**	**31.4**	**51.2**	**93.6**	**85.7**	**52.5**	**41.2**	**72.4**

Table 4: Exact matching (EM) accuracy by varying the levels of difficulty of the inference data on four benchmarks.

—

4. Final Design

- **System Architecture**:

  - The architecture comprises a frontend web app, backend services (Python and Flask), and connectors supporting MySQL, PostgreSQL, and SQLite.

- **User Interface Design**:

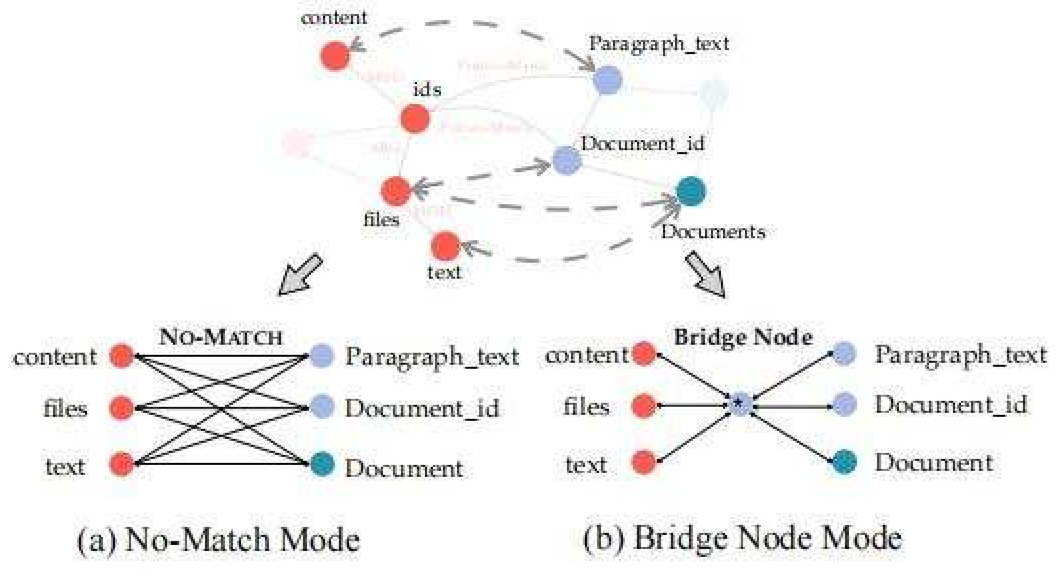
  - The interface includes a query input field, database selection, and query result display, with real-time validation and suggestions.

- **Backend Implementation**:

  - The backend, built with Flask and PyTorch, integrates Graphix-T5 for query parsing and database communication.

- **Testing and Validation**:

  - Extensive testing validated the system's performance, correctness, scalability, and usability under various conditions.



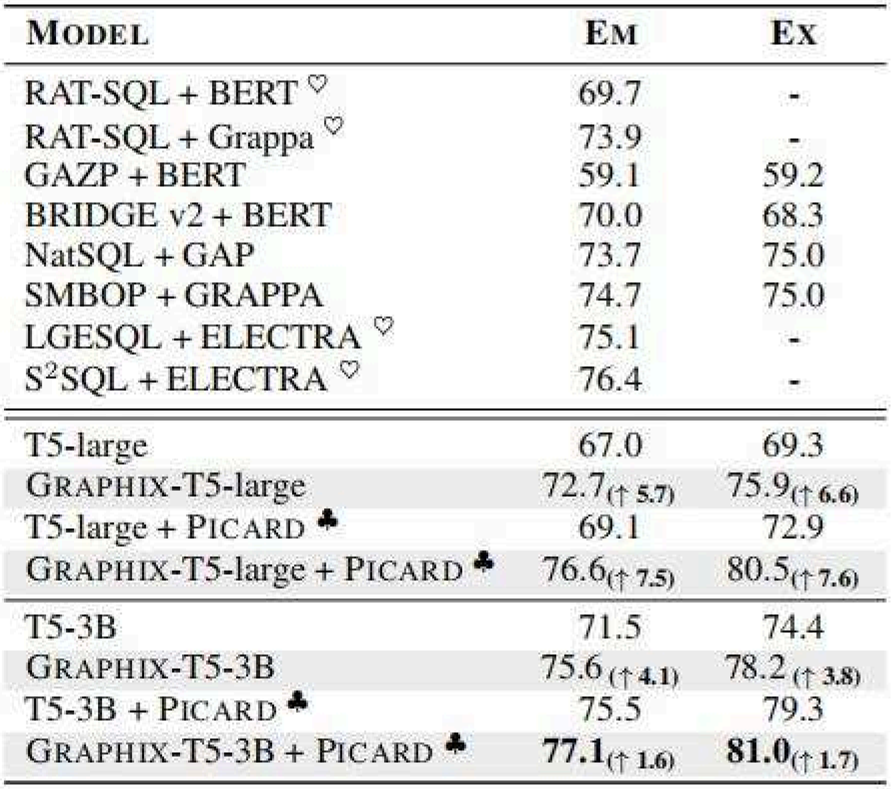
---

5. Implementation

- **Datasets and Benchmarks**: Testing was performed on datasets such as SPIDER, SYN, DK, and REALISTIC. Performance was measured using exact match (EM) and execution accuracy (EX).

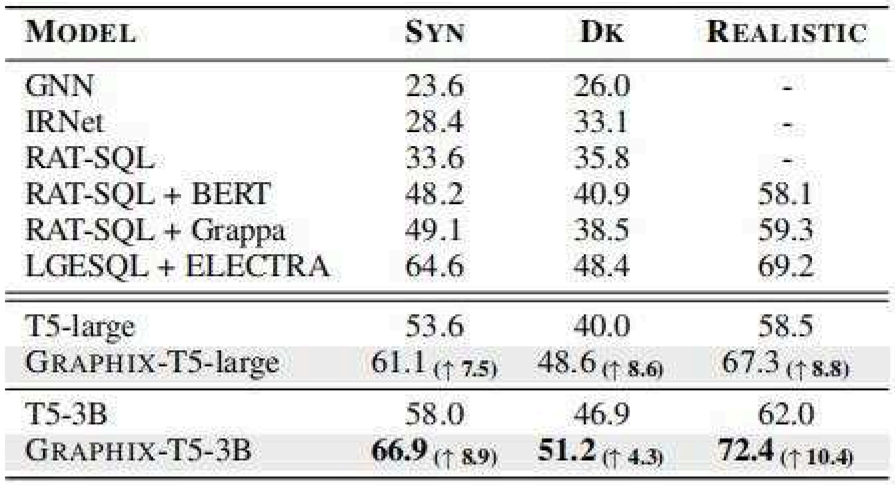
- **Development Process**: An agile approach was followed, with iterative design, development, and testing.

- **Tools and Technologies**: Frontend (React/Vue.js), backend (Python, Flask, PyTorch), databases (MySQL, PostgreSQL, SQLite), and cloud platforms (AWS, Heroku).



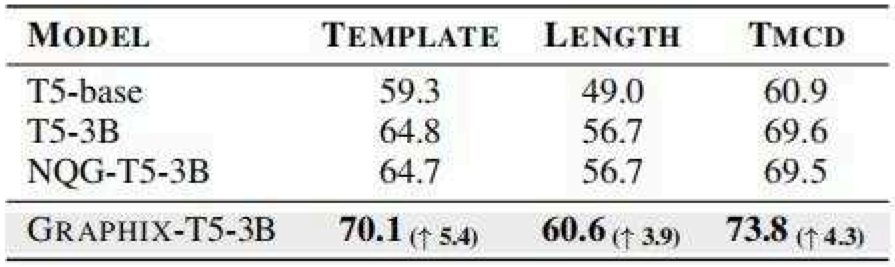
MODEL	EM	EX
RAT-SQL + BERT $^{\heartsuit}$	69.7	-
RAT-SQL + Grappa $^{\heartsuit}$	73.9	-
GAZP + BERT	59.1	59.2
BRIDGE v2 + BERT	70.0	68.3
NatSQL + GAP	73.7	75.0
SMBOP + GRAPPA	74.7	75.0
LGESQL + ELECTRA $^{\heartsuit}$	75.1	-
S^2SQL + ELECTRA $^{\heartsuit}$	76.4	-
T5-large	67.0	69.3
GRAPHIX-T5-large	$72.7_{(\uparrow 5.7)}$	$75.9_{(\uparrow 6.6)}$
T5-large + PICARD $^{\clubsuit}$	69.1	72.9
GRAPHIX-T5-large + PICARD $^{\clubsuit}$	$76.6_{(\uparrow 7.5)}$	$80.5_{(\uparrow 7.6)}$
T5-3B	71.5	74.4
GRAPHIX-T5-3B	$75.6_{(\uparrow 4.1)}$	$78.2_{(\uparrow 3.8)}$
T5-3B + PICARD $^{\clubsuit}$	75.5	79.3
GRAPHIX-T5-3B + PICARD $^{\clubsuit}$	**77.1**$_{(\uparrow 1.6)}$	**81.0**$_{(\uparrow 1.7)}$

Table 1: Exact match (EM) and execution (EX) accuracy (%) on SPIDER development set. $^{\heartsuit}$ means the model does not predict SQL values. $^{\clubsuit}$ means the model uses the constrained decoding PICARD. $\uparrow$ is an absolute improvement.



MODEL	SYN	DK	REALISTIC
GNN	23.6	26.0	-
IRNet	28.4	33.1	-
RAT-SQL	33.6	35.8	-
RAT-SQL + BERT	48.2	40.9	58.1
RAT-SQL + Grappa	49.1	38.5	59.3
LGESQL + ELECTRA	64.6	48.4	69.2
T5-large	53.6	40.0	58.5
GRAPHIX-T5-large	61.1 (↑ 7.5)	48.6 (↑ 8.6)	67.3 (↑ 8.8)
T5-3B	58.0	46.9	62.0
GRAPHIX-T5-3B	**66.9** (↑ 8.9)	**51.2** (↑ 4.3)	**72.4** (↑ 10.4)

Table 2: Exact match (EM) accuracy (%) on SYN, DK and REALISTIC benchmark.



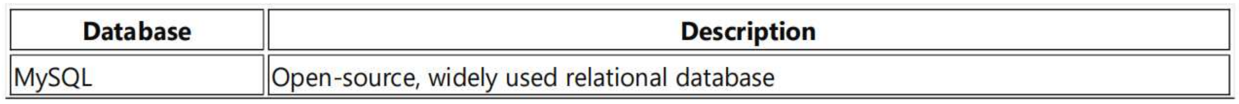
MODEL	TEMPLATE	LENGTH	TMCD
T5-base	59.3	49.0	60.9
T5-3B	64.8	56.7	69.6
NQG-T5-3B	64.7	56.7	69.5
GRAPHIX-T5-3B	**70.1** (↑ 5.4)	**60.6** (↑ 3.9)	**73.8** (↑ 4.3)

Table 3: Exact match (EM) accuracy (%) on compositional dataset SPIDER-SSP.

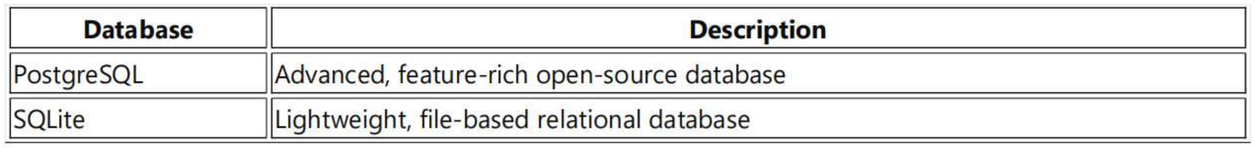
---

6. Data Integration

- **Data Sources**: Supports MySQL, PostgreSQL, and SQLite.



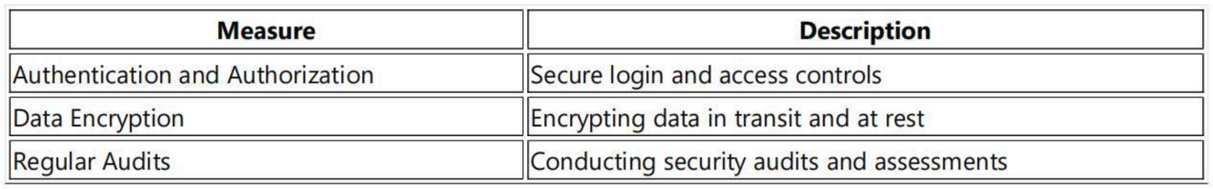
Database	Description
MySQL	Open-source, widely used relational database



Database	Description
PostgreSQL	Advanced, feature-rich open-source database
SQLite	Lightweight, file-based relational database

- **Data Handling**: Inputs are processed, converted into SQL queries, and results are retrieved and displayed to users.

- **Security and Privacy**: Ensures secure login, encryption, and regular audits.



Measure	Description
Authentication and Authorization	Secure login and access controls
Data Encryption	Encrypting data in transit and at rest
Regular Audits	Conducting security audits and assessments

---

7. User Interface and Experience

- **Frontend Development**: Built for usability and responsiveness with interactive elements.

- **User Experience**: Offers real-time feedback, clear visualizations, and query refinement options.

- **Accessibility**: Designed with keyboard navigation, screen reader support, and high-contrast color schemes.

---

8. Testing and Validation

- **Testing Plan**: Tested across various scenarios (simple, complex queries, performance, and usability.

- **Testing Results**: The model outperformed baselines on complex benchmarks like SPIDER and SYN, demonstrating high accuracy and performance.

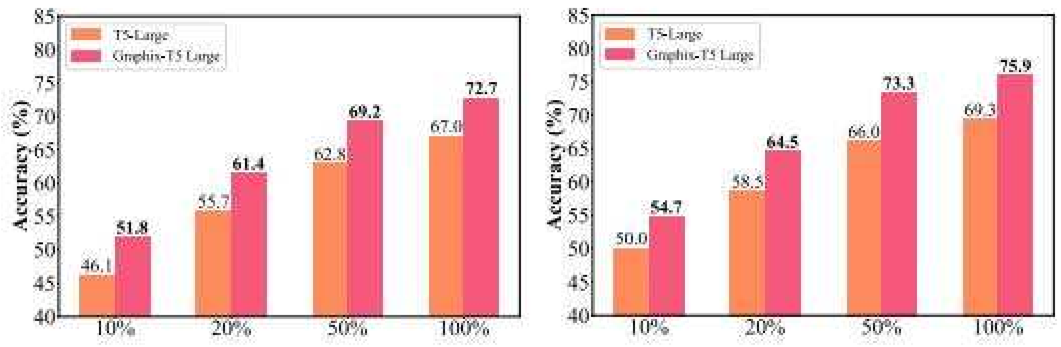


Figure 4: Exact match (EM) (left) and execution (EX) (right) accuracy (%) on SPIDER low-resource setting.

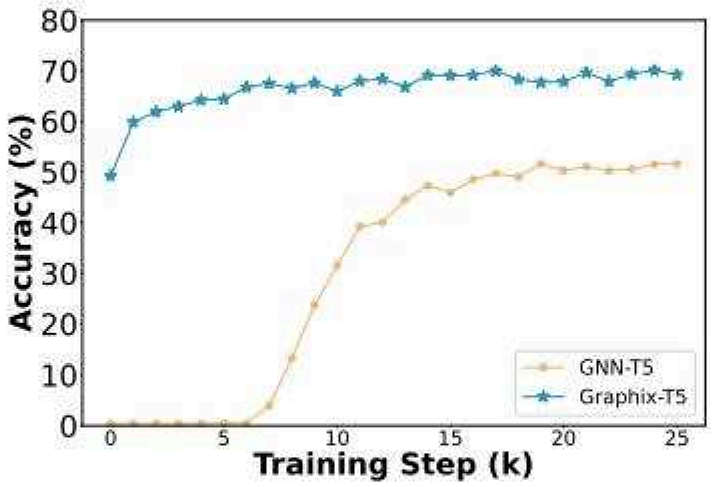


Figure 5: The performance of the validation sets during the convergence of GRAPHIX-T5 and GNN-T5 on SPIDER. It can be clearly demonstrated that GNN-T5 has extremely unsatisfactory performance, due to catastrophic forgetting.

---

9. Deployment

- **Deployment Strategy**: Deployed on cloud platforms, with auto-scaling and load balancing to handle increasing load.

- **Maintenance and Updates**: Regular updates for bug fixes, new features, and performance monitoring.

Step 2: Prototype Development

Small-Scale Implementat

The technical implementation of this service involves several components working together seamlessly. The core of the service lies in its ability to take natural language input, convert it into SQL queries, execute those queries against a database, and return the results in natural language. This involves the integration of Natural Language Processing (NLP) models, SQL databases, and a web-based interface for end-users.

The primary model for language processing will be Google Gemini or OpenAI GPT, which excels at converting natural language into SQL. The prototype will be built using Python for backend

Natural Language to SQL using Google's Gemini Pro | Python | Google AI Studio

Installation

In [1]:
```
!pip install -q google-generativeai==0.3.1
```
```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 146.6/146.6 kB 1.3 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 598.7/598.7 kB 4.5 MB/s eta 0:00:00
```

Imports

In [2]:
```
import google.generativeai as genai
from pathlib import Path
import sqlite3
```

Version

In [3]:
```
genai.__version__
```
Out[3]:
```
'0.3.1'
```

Secret Key

In [4]:
```
from google.colab import userdata

genai.configure(api_key = userdata.get('GEMINI_KEY'))
```

Configurations

In [5]:
```
# Set up the model
generation_config = {
  "temperature": 0.4,
  "top_p": 1,
  "top_k": 32,
  "max_output_tokens": 4096,
}

safety_settings = [
  {
    "category": "HARM_CATEGORY_HARASSMENT",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE"
  },
  {
    "category": "HARM_CATEGORY_HATE_SPEECH",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE"
  },
  {
    "category": "HARM_CATEGORY_SEXUALLY_EXPLICIT",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE"
  },
  {
    "category": "HARM_CATEGORY_DANGEROUS_CONTENT",
    "threshold": "BLOCK_MEDIUM_AND_ABOVE"
  }
]
```

Model Instance

In [6]:
```
model = genai.GenerativeModel(model_name = "gemini-pro",
                              generation_config = generation_config,
                              safety_settings = safety_settings)
```

SQL Query Executor

In [7]:
```
def read_sql_query(sql, db):
    conn = sqlite3.connect(db)
    cur = conn.cursor()
    cur.execute(sql)
    rows = cur.fetchall()
    for row in rows:
        print(row)
    conn.close()
```

In [10]:
```
import sqlite3
import pandas as pd

# Function to load a single CSV file into an SQLite table
def load_csv_to_sqlite(csv_file, table_name, db):
```

```python
    # Connect to the SQLite database
    conn = sqlite3.connect(db)
    # Load CSV into a pandas DataFrame
    df = pd.read_csv(csv_file)
    # Write the DataFrame to an SQLite table
    df.to_sql(table_name, conn, if_exists='replace', index=False)
    # Close the connection
    conn.close()

# Function to execute and print SQL query results
def read_sql_query(sql, db):
    conn = sqlite3.connect(db)
    cur = conn.cursor()
    cur.execute(sql)
    rows = cur.fetchall()
    for row in rows:
        print(row)
    conn.close()

# Load the CSV file into an SQLite table
load_csv_to_sqlite('mcdonalds.csv', 'table1', 'example.db')

# Write your SQL query
sql_query = '''
SELECT * FROM table1
'''

# Execute the query and print the results
read_sql_query(sql_query, 'example.db')
```

```
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', '-3', 61, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', '+2', 51, 'Every three months', 'Female')
('No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', '+1', 62, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', '+4', 69, 'Once a week', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', '+2', 49, 'Once a month', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+2', 55, 'Every three months', 'Male')
('Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', '+2', 56, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'I love it!+5', 23, 'Once a week', 'Female')
('No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'I hate it!-5', 58, 'Once a year', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', '+1', 32, 'Every three months', 'Female')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', '-2', 53, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', '+3', 28, 'Once a month', 'Male')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'No', '0', 65, 'Every three months', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'I love it!+5', 54, 'Once a month', 'Male')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', '-3', 67, 'Once a month', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+3', 51, 'Once a month', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '0', 34, 'Once a month', 'Female')
('Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No', '+2', 31, 'Once a month', 'Male')
('No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', '-4', 47, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'I love it!+5', 37, 'More than once a week', 'Mal
e')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', '+2', 41, 'Once a year', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', '+2', 36, 'Once a month', 'Male')
('Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', '-2', 23, 'Once a week', 'Female')
('Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'I love it!+5', 50, 'More than once a week', 'Femal
e')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+2', 39, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', '0', 35, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', '0', 20, 'Every three months', 'Male')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', '0', 34, 'Once a month', 'Female')
('No', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No', '0', 55, 'Every three months', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+3', 24, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+3', 44, 'Once a month', 'Male')
('No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+3', 40, 'Once a month', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+1', 55, 'Once a month', 'Male')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', '-4', 37, 'Never', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+4', 48, 'Once a week', 'Male')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', '-2', 54, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+3', 38, 'Every three months', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'I hate it!-5', 49, 'Every three months', 'Female')
('No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', '-4', 35, 'Never', 'Female')
('No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', '-4', 55, 'Once a year', 'Male')
('Yes', 'No', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', '+2', 31, 'Once a year', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+3', 57, 'More than once a week', 'Male')
('Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+4', 69, 'More than once a week', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', '+3', 37, 'Every three months', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'I hate it!-5', 60, 'Never', 'Male')
('Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', '+2', 36, 'Once a month', 'Male')
('No', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', '+1', 54, 'Every three months', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', '+4', 66, 'Once a month', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '0', 32, 'Once a month', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+2', 67, 'Once a year', 'Male')
('No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', '-4', 42, 'Once a week', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'No', '+2', 26, 'Once a month', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', '+1', 58, 'Once a year', 'Female')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'I hate it!-5', 48, 'Once a month', 'Male')
('Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+4', 54, 'Once a month', 'Female')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '0', 62, 'Once a year', 'Male')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', '-4', 52, 'Once a year', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'No', '0', 53, 'Every three months', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+4', 47, 'More than once a week', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'I love it!+5', 26, 'Once a month', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'I love it!+5', 29, 'Once a week', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+3', 24, 'Once a week', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'I love it!+5', 25, 'Once a week', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', 'No', '0', 54, 'Once a month', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+4', 23, 'Once a month', 'Female')
('No', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', '0', 34, 'Never', 'Female')
('No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', '+1', 47, 'Every three months', 'Male')
('Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', '+3', 22, 'Once a week', 'Male')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '0', 47, 'Once a year', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+4', 56, 'Once a month', 'Male')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', '-1', 55, 'Every three months', 'Male')
('Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+3', 56, 'Every three months', 'Female')
('No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', '0', 56, 'Every three months', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+2', 37, 'Every three months', 'Female')
('Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'I love it!+5', 50, 'Once a week', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', '-1', 40, 'Every three months', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+4', 61, 'Once a week', 'Male')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'No', '-2', 55, 'Every three months', 'Male')
('Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', '-1', 45, 'Once a year', 'Female')
('Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+1', 42, 'Once a month', 'Male')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'No', 'Yes', '+2', 52, 'Every three months', 'Male')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+2', 58, 'Once a year', 'Female')
('No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+1', 48, 'Once a month', 'Male')
('Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+3', 58, 'Once a week', 'Male')
('No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', '-3', 18, 'Once a year', 'Male')
('Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+4', 68, 'Once a month', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', '+1', 49, 'Once a month', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'I hate it!-5', 50, 'Once a week', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', '+2', 43, 'Every three months', 'Female')
('Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', '+4', 67, 'Every three months', 'Male')
```

```
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', '+3', 27, 'Once a month', 'Female')
('Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', '+1', 61, 'Once a month', 'Male')
('Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+3', 55, 'Once a month', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'I love it!+5', 29, 'Once a week', 'Male')
('Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+4', 31, 'Once a month', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', '+2', 50, 'Once a year', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', '-3', 55, 'Never', 'Male')
('No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', '+1', 18, 'Once a month', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+1', 52, 'Every three months', 'Female')
('No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'No', '0', 43, 'Once a month', 'Male')
('No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'I hate it!-5', 47, 'Once a year', 'Male')
('Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', '+2', 36, 'Once a week', 'Female')
('Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', '+3', 52, 'Once a month', 'Female')
('Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', '+4', 41, 'Every three months', 'Male')
('No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes', '-3', 30, 'Every three months', 'Male')
```

Define Prompt

In [14]:
```
prompt_parts_1 = [
    """You are an expert in converting English questions to SQL code! The SQL database has the name example.db and has the

    For example,
    Example 1 - How many people find McDonald's food yummy? The SQL command will be something like this:
    SELECT COUNT(*) FROM mcdonalds_data WHERE yummy = 1;

    Example 2 - How many people think McDonald's food is both cheap and tasty?
    SELECT COUNT(*) FROM mcdonalds_data WHERE cheap = 1 AND tasty = 1;

    Example 3 - What is the average age of people who visit McDonald's frequently?
    SELECT AVG(Age) FROM mcdonalds_data WHERE VisitFrequency = 'frequently';

    Do not include backticks (`) or \n in the output."""
]
```

In [16]:
```
question = "How many people think McDonald's food is both greasy and fast?"
```

In [17]:
```
prompt_parts = [prompt_parts_1[0], question]
response = model.generate_content(prompt_parts)
response.text
```

Out[17]:
```
'SELECT COUNT(*) FROM mcdonalds_data WHERE greasy = 1 AND fast = 1;'
```

In [19]:
```
read_sql_query("""SELECT COUNT(*) FROM table1 WHERE greasy = 1 AND fast = 1;
""",
               "example.db")
```

```
(0,)
```

Combine it into Function

In [26]:
```
import sqlite3
import pandas as pd

# Function to load CSV into SQLite table
def load_csv_to_sqlite(csv_file, table_name, db):
    # Connect to the SQLite database
    conn = sqlite3.connect(db)
    # Load CSV into a pandas DataFrame
    df = pd.read_csv(csv_file)
    # Write the DataFrame to an SQLite table
    df.to_sql(table_name, conn, if_exists='replace', index=False)
    # Close the connection
    conn.close()

# Load the CSV file into the 'mcdonalds_data' table
load_csv_to_sqlite('mcdonalds.csv', 'mcdonalds_data', 'example.db')
```

In [27]:
```
def generate_gemini_response(question, input_prompt):
    prompt_parts = [input_prompt, question]
    response = model.generate_content(prompt_parts)
    output = read_sql_query(response.text, "example.db")
    return output
```

In [28]:
```
generate_gemini_response("How many people like mcdonalds food but find it expensive?",
                         prompt_parts_1[0])
```

```
(0,)
```

In [48]:
```
import sqlite3
import pandas as pd

# Function to load the CSV file into the SQLite database
def load_csv_to_sqlite(csv_file, table_name, db):
    conn = sqlite3.connect(db)
    df = pd.read_csv(csv_file)
    df.to_sql(table_name, conn, if_exists='replace', index=False)
    conn.close()

# Function to generate SQL query, execute it, and print the results
def execute_query(question, input_prompt, db):
    # Generate the SQL query based on the question and prompt
    prompt_parts = [input_prompt, question]
    response = model.generate_content(prompt_parts)
    sql_query = response.text.strip()  # Ensure no leading/trailing whitespace

    # Print the SQL query for debugging purposes
    print(f"Generated SQL Query: {sql_query}")

    # Execute the SQL query and print the results
    conn = sqlite3.connect(db)
    cur = conn.cursor()
    try:
        cur.execute(sql_query)
        rows = cur.fetchall()
        if rows:
            for row in rows:
                print(row)
                print(row)
        else:
            print("No results found.")
    except sqlite3.OperationalError as e:
        print(f"SQL Error: {e}")
    finally:
        conn.close()

# Example usage
# Ensure to replace 'mcdonalds.csv' with your actual CSV file path and set 'model' properly
load_csv_to_sqlite('mcdonalds.csv', 'mcdonalds_data', 'example.db')

# Define your prompt for generating SQL queries
prompt_parts_1 = [
    '''You are an expert in converting English questions to SQL code! The SQL database has the name `example.db` and contair

- `yummy` (Yes/No)
- `convenient` (Yes/No)
- `spicy` (Yes/No)
- `fattening` (Yes/No)
- `greasy` (Yes/No)
- `fast` (Yes/No)
- `cheap` (Yes/No)
- `tasty` (Yes/No)
- `expensive` (Yes/No)
- `healthy` (Yes/No)
- `disgusting` (Yes/No)
- `Like` (numeric scale, e.g., -3)
- `Age` (integer)
- `VisitFrequency` (e.g., 'Every three months')
- `Gender` (e.g., 'Female')

For example,
Example 1 - How many people find McDonald's food tasty?
`SELECT COUNT(*) FROM mcdonalds_data WHERE tasty = 'Yes';`

Example 2 - What is the average age of people who think McDonald's food is both greasy and cheap?
`SELECT AVG(Age) FROM mcdonalds_data WHERE greasy = 'Yes' AND cheap = 'Yes';`

Example 3 - How many people who find McDonald's food both convenient and fast have a Like score greater than 0?
`SELECT COUNT(*) FROM mcdonalds_data WHERE convenient = 'Yes' AND fast = 'Yes' AND Like > 0;`

Example 4 - What percentage of people consider McDonald's food expensive but still find it tasty?
`SELECT (COUNT(*) * 100.0 / (SELECT COUNT(*) FROM mcdonalds_data)) AS percentage FROM mcdonalds_data WHERE expensive = 'Yes

Example 5 - What is the most common visit frequency among people who think McDonald's food is fattening?
`SELECT VisitFrequency FROM mcdonalds_data WHERE fattening = 'Yes' GROUP BY VisitFrequency ORDER BY COUNT(*) DESC LIMIT 1;`

Example 6 - How many of the first 10 people find McDonald's food yummy?
`SELECT COUNT(*) FROM (SELECT * FROM mcdonalds_data LIMIT 10) AS limited_data WHERE yummy = 'Yes';`
Example 7 - Determine the average age of customers who like healthy food but dislike greasy food, and also analyze their vi:
`SELECT
  AVG(Age) AS average_age,
  VisitFrequency,
  Gender
FROM mcdonalds_data
WHERE
  healthy = 'Yes' AND greasy = 'No'
GROUP BY
  VisitFrequency,
  Gender;`

Example 8 - Analyze customer segments based on their food preferences (e.g., "yummy," "spicy"), and demographic and behavior
    `WITH PreferenceSegments AS (
    SELECT
        Gender,
        VisitFrequency,
        Age,
        CASE
            WHEN yummy = 'Yes' AND healthy = 'Yes' THEN 'Yummy and Healthy'
            WHEN spicy = 'Yes' THEN 'Spicy Lovers'
            WHEN fast = 'Yes' THEN 'Fast Food Lovers'
            WHEN cheap = 'Yes' THEN 'Price Conscious'
            ELSE 'Other Preferences'
        END AS Food_Preference,
        CASE
            WHEN Like > 0 THEN 'Positive'
            WHEN Like < 0 THEN 'Negative'
            ELSE 'Neutral'
        END AS Like_Sentiment
    FROM
        your_table_name
    WHERE
        Gender IS NOT NULL
)
SELECT
    ps.Gender,
    ps.VisitFrequency,
    ps.Food_Preference,
    COUNT(*) AS Segment_Size,
    ROUND((COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (PARTITION BY ps.Gender)), 2) AS Segment_Percentage,
    RANK() OVER (PARTITION BY ps.Gender ORDER BY COUNT(*) DESC) AS Rank_Within_Gender,
    (CASE
         WHEN AVG(ps.Age) > 40 THEN 'Older Age Group'
         ELSE 'Younger Age Group'
     END) AS Age_Group,
    ps.Like_Sentiment
FROM
    PreferenceSegments ps
GROUP BY
    ps.Gender,
    ps.VisitFrequency,
    ps.Food_Preference,
    ps.Like_Sentiment
HAVING
    Segment_Size > 5  -- Exclude segments with fewer than 5 customers
ORDER BY
    ps.Gender,
    Rank_Within_Gender;`

Do not include backticks (`) or `\n` in the output.'''
]

# Example questions
questions = [
    "Determine the average age of customers who like healthy food but dislike greasy food, and also analyze their visit fre
    "Analyze customer segments based on their food preferences (e.g., 'yummy,' 'spicy'), and demographic and behavioral fact
    "perform segmentation analysis"
    ]
```

```python
# Execute and print results for each random question
for question in questions:
    print(f"\nQuestion: {question}")
    execute_query(question, prompt_parts_1[0], 'example.db')
```

```
Question: Determine the average age of customers who like healthy food but dislike greasy food, and also analyze their visit
frequency and gender distribution.
Generated SQL Query: SELECT
  AVG(Age) AS average_age,
  VisitFrequency,
  Gender
FROM mcdonalds_data
WHERE
  healthy = 'Yes' AND greasy = 'No'
GROUP BY
  VisitFrequency,
  Gender;
(52.61290322580645, 'Every three months', 'Female')
(53.92307692307692, 'Every three months', 'Male')
(40.5, 'More than once a week', 'Female')
(45.375, 'More than once a week', 'Male')
(55.0, 'Never', 'Female')
(47.191489361702125, 'Once a month', 'Female')
(46.96, 'Once a month', 'Male')
(43.55, 'Once a week', 'Female')
(44.083333333333336, 'Once a week', 'Male')
(53.333333333333336, 'Once a year', 'Female')
(59.77777777777778, 'Once a year', 'Male')

Question: Analyze customer segments based on their food preferences (e.g., 'yummy,' 'spicy'), and demographic and behavioral
factors like age, visit frequency, and gender.
Generated SQL Query: WITH PreferenceSegments AS (
    SELECT
        Gender,
        VisitFrequency,
        Age,
        CASE
            WHEN yummy = 'Yes' AND healthy = 'Yes' THEN 'Yummy and Healthy'
            WHEN spicy = 'Yes' THEN 'Spicy Lovers'
            WHEN fast = 'Yes' THEN 'Fast Food Lovers'
            WHEN cheap = 'Yes' THEN 'Price Conscious'
            ELSE 'Other Preferences'
        END AS Food_Preference,
        CASE
            WHEN Like > 0 THEN 'Positive'
            WHEN Like < 0 THEN 'Negative'
            ELSE 'Neutral'
        END AS Like_Sentiment
    FROM
        mcdonalds_data
    WHERE
        Gender IS NOT NULL
)
SELECT
    ps.Gender,
    ps.VisitFrequency,
    ps.Food_Preference,
    COUNT(*) AS Segment_Size,
    ROUND((COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (PARTITION BY ps.Gender)), 2) AS Segment_Percentage,
    RANK() OVER (PARTITION BY ps.Gender ORDER BY COUNT(*) DESC) AS Rank_Within_Gender,
    (CASE
         WHEN AVG(ps.Age) > 40 THEN 'Older Age Group'
         ELSE 'Younger Age Group'
     END) AS Age_Group,
    ps.Like_Sentiment
FROM
    PreferenceSegments ps
GROUP BY
    ps.Gender,
    ps.VisitFrequency,
    ps.Food_Preference,
    ps.Like_Sentiment
HAVING
    Segment_Size > 5  -- Exclude segments with fewer than 5 customers
ORDER BY
    ps.Gender,
    Rank_Within_Gender;
('Female', 'Once a month', 'Fast Food Lovers', 126, 17.57, 1, 'Older Age Group', 'Negative')
('Female', 'Every three months', 'Fast Food Lovers', 99, 13.81, 2, 'Older Age Group', 'Negative')
('Female', 'Once a year', 'Fast Food Lovers', 87, 12.13, 3, 'Older Age Group', 'Negative')
('Female', 'Once a week', 'Fast Food Lovers', 55, 7.67, 4, 'Younger Age Group', 'Negative')
('Female', 'Once a month', 'Yummy and Healthy', 41, 5.72, 5, 'Older Age Group', 'Negative')
('Female', 'Every three months', 'Yummy and Healthy', 31, 4.32, 6, 'Older Age Group', 'Negative')
('Female', 'Never', 'Fast Food Lovers', 30, 4.18, 7, 'Older Age Group', 'Positive')
('Female', 'Once a year', 'Fast Food Lovers', 21, 2.93, 8, 'Older Age Group', 'Neutral')
('Female', 'Once a week', 'Fast Food Lovers', 20, 2.79, 9, 'Younger Age Group', 'Positive')
('Female', 'Once a month', 'Fast Food Lovers', 18, 2.51, 10, 'Younger Age Group', 'Positive')
('Female', 'Once a month', 'Fast Food Lovers', 17, 2.37, 11, 'Older Age Group', 'Neutral')
('Female', 'Once a week', 'Yummy and Healthy', 17, 2.37, 11, 'Older Age Group', 'Positive')
('Female', 'Every three months', 'Fast Food Lovers', 16, 2.23, 13, 'Older Age Group', 'Neutral')
('Female', 'Once a month', 'Yummy and Healthy', 14, 1.95, 14, 'Older Age Group', 'Positive')
('Female', 'Once a week', 'Yummy and Healthy', 13, 1.81, 15, 'Older Age Group', 'Negative')
('Female', 'Never', 'Fast Food Lovers', 12, 1.67, 16, 'Older Age Group', 'Negative')
('Female', 'Once a year', 'Fast Food Lovers', 12, 1.67, 16, 'Older Age Group', 'Positive')
('Female', 'Every three months', 'Spicy Lovers', 10, 1.39, 18, 'Older Age Group', 'Negative')
('Female', 'Once a month', 'Spicy Lovers', 9, 1.26, 19, 'Older Age Group', 'Negative')
('Female', 'Every three months', 'Fast Food Lovers', 8, 1.12, 20, 'Older Age Group', 'Positive')
('Female', 'Never', 'Fast Food Lovers', 8, 1.12, 20, 'Older Age Group', 'Neutral')
('Female', 'Never', 'Other Preferences', 8, 1.12, 20, 'Older Age Group', 'Positive')
('Female', 'Once a year', 'Spicy Lovers', 8, 1.12, 20, 'Older Age Group', 'Negative')
('Female', 'More than once a week', 'Fast Food Lovers', 7, 0.98, 24, 'Younger Age Group', 'Positive')
('Female', 'Every three months', 'Other Preferences', 6, 0.84, 25, 'Older Age Group', 'Negative')
('Female', 'More than once a week', 'Fast Food Lovers', 6, 0.84, 25, 'Older Age Group', 'Negative')
('Female', 'More than once a week', 'Yummy and Healthy', 6, 0.84, 25, 'Younger Age Group', 'Negative')
('Female', 'More than once a week', 'Yummy and Healthy', 6, 0.84, 25, 'Younger Age Group', 'Positive')
('Female', 'Once a week', 'Spicy Lovers', 6, 0.84, 25, 'Older Age Group', 'Negative')
('Male', 'Once a month', 'Fast Food Lovers', 117, 19.93, 1, 'Older Age Group', 'Negative')
('Male', 'Every three months', 'Fast Food Lovers', 73, 12.44, 2, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Fast Food Lovers', 55, 9.37, 3, 'Younger Age Group', 'Negative')
('Male', 'Once a year', 'Fast Food Lovers', 45, 7.67, 4, 'Older Age Group', 'Negative')
('Male', 'Never', 'Fast Food Lovers', 35, 5.96, 5, 'Older Age Group', 'Positive')
('Male', 'Every three months', 'Fast Food Lovers', 31, 5.28, 6, 'Older Age Group', 'Neutral')
('Male', 'Once a month', 'Yummy and Healthy', 26, 4.43, 7, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Yummy and Healthy', 20, 3.41, 8, 'Older Age Group', 'Negative')
('Male', 'Once a year', 'Fast Food Lovers', 16, 2.73, 9, 'Older Age Group', 'Positive')
('Male', 'Every three months', 'Other Preferences', 14, 2.39, 10, 'Older Age Group', 'Negative')
('Male', 'Once a year', 'Fast Food Lovers', 13, 2.21, 11, 'Older Age Group', 'Neutral')
('Male', 'Never', 'Fast Food Lovers', 12, 2.04, 12, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Yummy and Healthy', 12, 2.04, 12, 'Younger Age Group', 'Positive')
('Male', 'More than once a week', 'Fast Food Lovers', 11, 1.87, 14, 'Older Age Group', 'Negative')
('Male', 'Once a month', 'Fast Food Lovers', 11, 1.87, 14, 'Older Age Group', 'Neutral')
```

```
('Male', 'Once a month', 'Fast Food Lovers', 11, 1.87, 14, 'Younger Age Group', 'Positive')
('Male', 'Every three months', 'Spicy Lovers', 10, 1.7, 17, 'Older Age Group', 'Negative')
('Male', 'Every three months', 'Yummy and Healthy', 10, 1.7, 17, 'Older Age Group', 'Negative')
('Male', 'Once a month', 'Spicy Lovers', 10, 1.7, 17, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Fast Food Lovers', 8, 1.36, 20, 'Younger Age Group', 'Positive')
('Male', 'Every three months', 'Fast Food Lovers', 7, 1.19, 21, 'Younger Age Group', 'Positive')
('Male', 'More than once a week', 'Yummy and Healthy', 7, 1.19, 21, 'Younger Age Group', 'Positive')
('Male', 'Once a month', 'Other Preferences', 7, 1.19, 21, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Fast Food Lovers', 7, 1.19, 21, 'Younger Age Group', 'Neutral')
('Male', 'Once a year', 'Other Preferences', 7, 1.19, 21, 'Older Age Group', 'Positive')
('Male', 'Once a month', 'Yummy and Healthy', 6, 1.02, 26, 'Older Age Group', 'Positive')
('Male', 'Once a week', 'Other Preferences', 6, 1.02, 26, 'Younger Age Group', 'Negative')

Question: perform segmentation analysis
Generated SQL Query: WITH PreferenceSegments AS (
    SELECT
        Gender,
        VisitFrequency,
        Age,
        CASE
            WHEN yummy = 'Yes' AND healthy = 'Yes' THEN 'Yummy and Healthy'
            WHEN spicy = 'Yes' THEN 'Spicy Lovers'
            WHEN fast = 'Yes' THEN 'Fast Food Lovers'
            WHEN cheap = 'Yes' THEN 'Price Conscious'
            ELSE 'Other Preferences'
        END AS Food_Preference,
        CASE
            WHEN Like > 0 THEN 'Positive'
            WHEN Like < 0 THEN 'Negative'
            ELSE 'Neutral'
        END AS Like_Sentiment
    FROM
        mcdonalds_data
    WHERE
        Gender IS NOT NULL
)
SELECT
    ps.Gender,
    ps.VisitFrequency,
    ps.Food_Preference,
    COUNT(*) AS Segment_Size,
    ROUND((COUNT(*) * 100.0 / SUM(COUNT(*)) OVER (PARTITION BY ps.Gender)), 2) AS Segment_Percentage,
    RANK() OVER (PARTITION BY ps.Gender ORDER BY COUNT(*) DESC) AS Rank_Within_Gender,
    (CASE
         WHEN AVG(ps.Age) > 40 THEN 'Older Age Group'
         ELSE 'Younger Age Group'
     END) AS Age_Group,
    ps.Like_Sentiment
FROM
    PreferenceSegments ps
GROUP BY
    ps.Gender,
    ps.VisitFrequency,
    ps.Food_Preference,
    ps.Like_Sentiment
HAVING
    Segment_Size > 5  -- Exclude segments with fewer than 5 customers
ORDER BY
    ps.Gender,
    Rank_Within_Gender;
('Female', 'Once a month', 'Fast Food Lovers', 126, 17.57, 1, 'Older Age Group', 'Negative')
('Female', 'Every three months', 'Fast Food Lovers', 99, 13.81, 2, 'Older Age Group', 'Negative')
('Female', 'Once a year', 'Fast Food Lovers', 87, 12.13, 3, 'Older Age Group', 'Negative')
('Female', 'Once a week', 'Fast Food Lovers', 55, 7.67, 4, 'Younger Age Group', 'Negative')
('Female', 'Once a month', 'Yummy and Healthy', 41, 5.72, 5, 'Older Age Group', 'Negative')
('Female', 'Every three months', 'Yummy and Healthy', 31, 4.32, 6, 'Older Age Group', 'Negative')
('Female', 'Never', 'Fast Food Lovers', 30, 4.18, 7, 'Older Age Group', 'Positive')
('Female', 'Once a year', 'Fast Food Lovers', 21, 2.93, 8, 'Older Age Group', 'Neutral')
('Female', 'Once a week', 'Fast Food Lovers', 20, 2.79, 9, 'Younger Age Group', 'Positive')
('Female', 'Once a month', 'Fast Food Lovers', 18, 2.51, 10, 'Younger Age Group', 'Positive')
('Female', 'Once a month', 'Fast Food Lovers', 17, 2.37, 11, 'Older Age Group', 'Neutral')
('Female', 'Once a week', 'Yummy and Healthy', 17, 2.37, 11, 'Older Age Group', 'Positive')
('Female', 'Every three months', 'Fast Food Lovers', 16, 2.23, 13, 'Older Age Group', 'Neutral')
('Female', 'Once a month', 'Yummy and Healthy', 14, 1.95, 14, 'Older Age Group', 'Positive')
('Female', 'Once a week', 'Yummy and Healthy', 13, 1.81, 15, 'Older Age Group', 'Negative')
('Female', 'Never', 'Fast Food Lovers', 12, 1.67, 16, 'Older Age Group', 'Negative')
('Female', 'Once a year', 'Fast Food Lovers', 12, 1.67, 16, 'Older Age Group', 'Positive')
('Female', 'Every three months', 'Spicy Lovers', 10, 1.39, 18, 'Older Age Group', 'Negative')
('Female', 'Once a month', 'Spicy Lovers', 9, 1.26, 19, 'Older Age Group', 'Negative')
('Female', 'Every three months', 'Fast Food Lovers', 8, 1.12, 20, 'Older Age Group', 'Positive')
('Female', 'Never', 'Fast Food Lovers', 8, 1.12, 20, 'Older Age Group', 'Neutral')
('Female', 'Never', 'Other Preferences', 8, 1.12, 20, 'Older Age Group', 'Positive')
('Female', 'Once a year', 'Spicy Lovers', 8, 1.12, 20, 'Older Age Group', 'Negative')
('Female', 'More than once a week', 'Fast Food Lovers', 7, 0.98, 24, 'Younger Age Group', 'Positive')
('Female', 'Every three months', 'Other Preferences', 6, 0.84, 25, 'Older Age Group', 'Negative')
('Female', 'More than once a week', 'Fast Food Lovers', 6, 0.84, 25, 'Older Age Group', 'Negative')
('Female', 'More than once a week', 'Yummy and Healthy', 6, 0.84, 25, 'Younger Age Group', 'Negative')
('Female', 'More than once a week', 'Yummy and Healthy', 6, 0.84, 25, 'Younger Age Group', 'Positive')
('Female', 'Once a week', 'Spicy Lovers', 6, 0.84, 25, 'Older Age Group', 'Negative')
('Male', 'Once a month', 'Fast Food Lovers', 117, 19.93, 1, 'Older Age Group', 'Negative')
('Male', 'Every three months', 'Fast Food Lovers', 73, 12.44, 2, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Fast Food Lovers', 55, 9.37, 3, 'Younger Age Group', 'Negative')
('Male', 'Once a year', 'Fast Food Lovers', 45, 7.67, 4, 'Older Age Group', 'Negative')
('Male', 'Never', 'Fast Food Lovers', 35, 5.96, 5, 'Older Age Group', 'Positive')
('Male', 'Every three months', 'Fast Food Lovers', 31, 5.28, 6, 'Older Age Group', 'Neutral')
('Male', 'Once a month', 'Yummy and Healthy', 26, 4.43, 7, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Yummy and Healthy', 20, 3.41, 8, 'Older Age Group', 'Negative')
('Male', 'Once a year', 'Fast Food Lovers', 16, 2.73, 9, 'Older Age Group', 'Positive')
('Male', 'Every three months', 'Other Preferences', 14, 2.39, 10, 'Older Age Group', 'Negative')
('Male', 'Once a year', 'Fast Food Lovers', 13, 2.21, 11, 'Older Age Group', 'Neutral')
('Male', 'Never', 'Fast Food Lovers', 12, 2.04, 12, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Yummy and Healthy', 12, 2.04, 12, 'Younger Age Group', 'Positive')
('Male', 'More than once a week', 'Fast Food Lovers', 11, 1.87, 14, 'Older Age Group', 'Negative')
('Male', 'Once a month', 'Fast Food Lovers', 11, 1.87, 14, 'Older Age Group', 'Neutral')
('Male', 'Once a month', 'Fast Food Lovers', 11, 1.87, 14, 'Younger Age Group', 'Positive')
('Male', 'Every three months', 'Spicy Lovers', 10, 1.7, 17, 'Older Age Group', 'Negative')
('Male', 'Every three months', 'Yummy and Healthy', 10, 1.7, 17, 'Older Age Group', 'Negative')
('Male', 'Once a month', 'Spicy Lovers', 10, 1.7, 17, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Fast Food Lovers', 8, 1.36, 20, 'Younger Age Group', 'Positive')
('Male', 'Every three months', 'Fast Food Lovers', 7, 1.19, 21, 'Younger Age Group', 'Positive')
('Male', 'More than once a week', 'Yummy and Healthy', 7, 1.19, 21, 'Younger Age Group', 'Positive')
('Male', 'Once a month', 'Other Preferences', 7, 1.19, 21, 'Older Age Group', 'Negative')
('Male', 'Once a week', 'Fast Food Lovers', 7, 1.19, 21, 'Younger Age Group', 'Neutral')
('Male', 'Once a year', 'Other Preferences', 7, 1.19, 21, 'Older Age Group', 'Positive')
('Male', 'Once a month', 'Yummy and Healthy', 6, 1.02, 26, 'Older Age Group', 'Positive')
('Male', 'Once a week', 'Other Preferences', 6, 1.02, 26, 'Younger Age Group', 'Negative')
```

```
In [ ]:
```

```
In [ ]:
```

processing, with SQLite as the database for initial testing. As the product scales, we can expand support to more complex databases such as MySQL or PostgreSQL.

Key technical components include:

1. **NLP Module**: This module processes user inputs, identifies keywords and intent, and converts them into SQL queries using the generative AI model.
2. **SQL Query Executor**: Once the SQL query is generated, it is executed against the connected database (in the prototype stage, this will be SQLite). The results are fetched and formatted for the user.
3. **Natural Language Output**: The results of the SQL query are converted back into a human-readable form using the same NLP engine, ensuring that users do not have to interpret raw data.
4. **Web Interface**: A simple web interface will allow users to input their questions, view the results, and interact with the tool. This interface will be designed for non-technical users, focusing on ease of use and accessibility.

The implementation is modular, allowing each component to be developed and tested separately. For example, the SQL generator can be tested independently using various sample questions, while the interface can be developed using HTML, CSS, and JavaScript, with Flask or Django as the web framework.

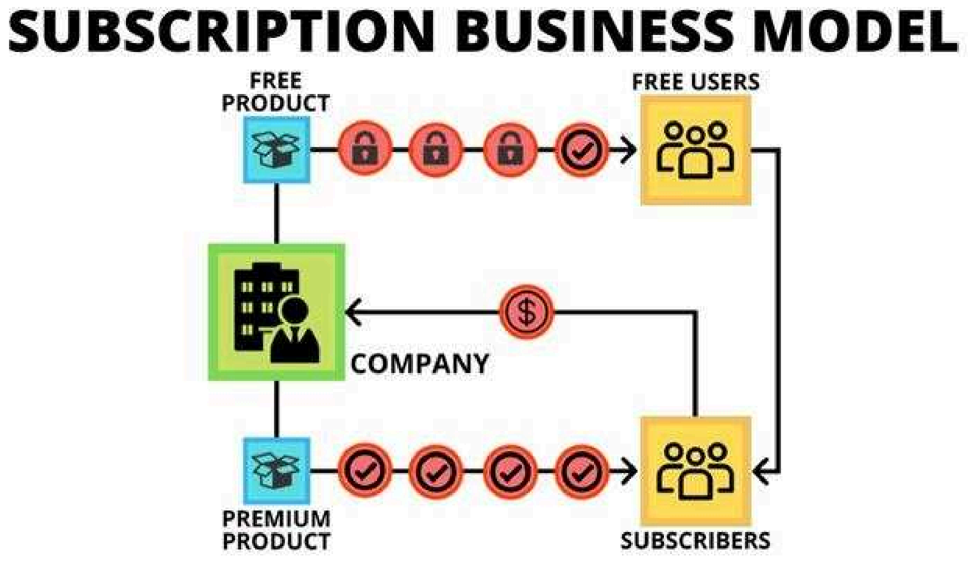
This modular approach ensures scalability and flexibility, allowing for the addition of new features such as support for more complex SQL queries (e.g., subqueries, joins) or integration with cloud databases like Amazon RDS or Google Cloud SQL.

Step 3: Business Modelling

Expanded Revenue Model: Subscription-Based and Enterprise Solutions

The primary business model for this service will be **subscription-based**, with multiple pricing tiers to accommodate different user needs. By offering different levels of access, we can target a wide variety of customers, from small business owners and freelancers to larger enterprises.

- **Freemium Model**: The free version of the product will allow users to perform a limited number of queries per month. This will encourage user engagement and increase customer retention. As users become familiar with the tool and recognize its value, we will push for conversions to the paid plans.
- **Basic Plan**: For a fixed monthly fee, users can access a larger number of queries per month, with standard SQL functionality. This plan will be targeted at small businesses or professionals who need moderate access to databases without hiring SQL experts.
- **Pro Plan**: The professional plan will include additional features such as advanced SQL querying, database analytics, and personalized customer support. This plan will be aimed at medium-sized businesses with more substantial data requirements.
- **Enterprise Solutions**: For larger organizations, we will offer customizable plans that include tailored solutions for internal database querying, white-label solutions for companies that want to integrate the service into their own platforms, and volume discounts for high query demands. Enterprises will also benefit from advanced analytics, multi-database support, and private cloud hosting options to meet their security needs.



Customer Segments:

- **SMEs and Startups**: These companies often have large amounts of data but lack the technical staff to extract actionable insights. Our service will empower these businesses by allowing them to query their databases without hiring data analysts.
- **Educational Institutions**: Universities and colleges could adopt the service as a learning tool, allowing students to practice querying databases without needing to learn SQL. This opens up new monetization possibilities, such as bulk licensing deals with educational institutions.
- **Freelancers and Consultants**: Data consultants and freelancers who handle database-related projects can use this tool to simplify their work and reduce the time spent on writing SQL queries, thus improving their efficiency and productivity.

Step 4: Financial Modelling

Market and Data Collection

The market for data querying and AI-based services in India is rapidly growing, driven by the country's expanding tech sector and the rising importance of data analytics across industries. The product will be launched in key sectors such as finance, healthcare, retail, and education, where database access is critical for decision-making.

Sources of Market Data:

- **IBEF Reports**: Provide insights into the growing need for data-driven solutions in sectors such as retail, healthcare, and education in India.
- **Government Reports**: Data on the rise of data literacy and technology adoption in small and medium enterprises across India can inform market penetration strategies.

Financial Forecasting in Indian Rupees

Assuming a **linear market growth** model, the financial equation will be as follows:

$y=mx(t)+cy = mx(t) + cy=mx(t)+c$

Where:

- **y** = Total Profit in INR
- **m** = Pricing of the service (₹5,000 per month for basic plan)
- **x(t)** = Number of paying users over time (starting with 200 users, growing at 7% per month)
- **c** = Fixed costs, including server maintenance, staff salaries, and infrastructure (approximately ₹2,00,000 per month)

Sample Calculation:

- Initial user base: **200 users**
- Monthly subscription fee: **₹5,000**
- Monthly user growth: **7%**

After **12 months**, the projected number of users is:

200×(1+0.07)12≈450200 \times (1 + 0.07)^{12} \approx 450200×(1+0.07)12≈450

Projected total revenue after 12 months:

450×5,000=₹22,50,000450 \times 5,000 = ₹22,50,000450×5,000=₹22,50,000

Conclusion

This expanded proposal outlines the development of a web-based service for converting natural language into SQL queries. The product addresses a critical market need by making database access easy for non-technical users, ensuring that businesses can leverage their data without additional SQL training. The project is feasible, viable, and highly monetizable through a subscription-based revenue model with potential enterprise solutions. By catering to key sectors like finance, healthcare, and education, the product has strong growth potential in the Indian market.