

# Text-to-SQL: Bridging the Gap Between Human Language and Databases

Text-to-SQL, also known as Natural Language to SQL (NL2SQL), is a rapidly evolving technology that translates natural, everyday language into Structured Query Language (SQL) commands. This innovative approach empowers users to interact with and retrieve data from databases simply by asking questions in plain English, eliminating the need for specialized knowledge of complex SQL syntax.

At its core, Text-to-SQL acts as an intelligent translator. It leverages the power of artificial intelligence, particularly **Natural Language Processing (NLP)** and sophisticated **AI models**, to understand the user's intent and generate the corresponding SQL query. This process allows individuals without a technical background to explore and analyze data, thereby democratizing data access within an organization.

## Retrieve data

Loading the data to create ecommerce database

Loading the customer dataset

% Total	% Received	% Xferd	Average Speed		Time	Time	Time	Current
			Dload	Upload	Total	Spent	Left	Speed
100 99461	0 99461	0 0	53222	0	--:--:--	0:00:01	--:--:--	53216

Loading the customer product dataset

% Total	% Received	% Xferd	Average Speed		Time	Time	Time	Current
			Dload	Upload	Total	Spent	Left	Speed
100 680k	0 680k	0 0	235k	0	--:--:--	0:00:02	--:--:--	235k

Loading the orders dataset

% Total	% Received	% Xferd	Average Speed		Time	Time	Time	Current
			Dload	Upload	Total	Spent	Left	Speed
100 230k	0 230k	0 0	26412	0	--:--:--	0:00:08	--:--:--	40087

## Setting up the database

```
import sqlite3
import pandas as pd
import os
```

## Define SQL schemas for creating tables

```
customers_schema = """ CREATE TABLE IF NOT EXISTS customers ( customer_id INT PRIMARY KEY,
first_name VARCHAR(50), last_name VARCHAR(50), email VARCHAR(50), phone_number VARCHAR(50),
address VARCHAR(50), city VARCHAR(50), country VARCHAR(50), postal_code VARCHAR(50),
loyalty_points INT ); """
```

```
products_schema = """ CREATE TABLE IF NOT EXISTS products ( product_id INT PRIMARY KEY,
product_name TEXT, description TEXT, price DECIMAL(10,2), discount_percentage DECIMAL(5,2), category
VARCHAR(50), brand TEXT, stock_quantity INT, color VARCHAR(50), size VARCHAR(20), weight
DECIMAL(5,2), dimensions TEXT, release_date DATE, rating DECIMAL(3,1), reviews_count INT, seller_name
TEXT, seller_rating DECIMAL(3,1), seller_reviews_count INT, shipping_method VARCHAR(20),
shipping_cost DECIMAL(6,2) ); """
```

```
orders_schema = """ CREATE TABLE IF NOT EXISTS orders ( order_id INT PRIMARY KEY, customer_id INT,
product_id INT, quantity INT, unit_price DECIMAL(10,2), total_price DECIMAL(10,2), order_date DATE,
shipping_address VARCHAR(255), payment_method VARCHAR(20), status VARCHAR(20), FOREIGN KEY
(customer_id) REFERENCES customers(customer_id), FOREIGN KEY (product_id) REFERENCES
products(product_id) ); """
```

**Note: The SQL scripts are AI generated**

## Data loading approach:

Initialize connection to None

Establish a connection to the SQLite database Create tables

Load data from CSV files into the tables using pandas

Read the CSV file into a pandas DataFrame

1. Get the expected schema for the current table
2. Handle missing/extra columns. And drop columns from DataFrame that are not in the schema
3. Reorder columns to match the defined schema exactly
4. Enforce data types

Commit the changes to the database

Close the connection if it was established

Database "ecommerce.db" created and Connected successfully.

Tables 'customers', 'products', and 'orders' created successfully.

Processing '/content/customers.csv' for table 'customers'...

-> Data from '/content/customers.csv' loaded into 'customers' table successfully.

Processing '/content/products.csv' for table 'products'...

-> Data from '/content/products.csv' loaded into 'products' table successfully.

Processing '/content/orders.csv' for table 'orders'...

-> Data from '/content/orders.csv' loaded into 'orders' table successfully.

Changes committed to the database successfully.

Connection to the database closed.

## Setting up the LLM Model

I am using google-generativeai package, the official Python SDK for the Gemini API.

Google genAI and google user data is used to established the connection with google LLM

I am going to use "gemini-2.5-flash" for this model

**Creating the prompt for LLM to understand the data structure and the data within the database. Convert the user questions written in plain English into accurate , efficient and syntactically correct SQL query based on a fixed database**

```
prompt = ""
```

## ROLE

You are an expert-level SQLite Database Engineer specializing in Natural Language to SQL (NL2SQL) translation. Your sole function is to convert user questions written in plain English into accurate, efficient, and syntactically correct SQLite queries based on a fixed database schema.

---

# CONTEXT

You are the core translation engine for a business intelligence dashboard. This tool allows non-technical employees to query the company's e-commerce database using natural language. The database dialect is always **SQLite**. Your responses will be executed directly on the database.

The database consists of the following three tables:

## customers table:

```
CREATE TABLE customers (  
  customer_id INT PRIMARY KEY,  
  first_name VARCHAR(50),  
  last_name VARCHAR(50),  
  email VARCHAR(50),  
  phone_number VARCHAR(50),  
  address VARCHAR(50),  
  city VARCHAR(50),  
  country VARCHAR(50),  
  postal_code VARCHAR(50),  
  loyalty_points INT  
);
```

## products table:

```
CREATE TABLE products (  
  product_id INT PRIMARY KEY,  
  product_name TEXT,  
  description TEXT,  
  price DECIMAL(10,2),  
  discount_percentage DECIMAL(5,2),  
  category VARCHAR(50),  
  brand TEXT,  
  stock_quantity INT,  
  color VARCHAR(50),  
  size VARCHAR(20),  
  weight DECIMAL(5,2),  
  dimensions TEXT,  
  release_date DATE,  
  rating DECIMAL(3,1),  
  reviews_count INT,  
  seller_name TEXT,  
  seller_rating DECIMAL(3,1),  
  seller_reviews_count INT,  
  shipping_method VARCHAR(20),  
  shipping_cost DECIMAL(6,2)  
);
```

## orders table:

```
CREATE TABLE orders (  
  order_id INT PRIMARY KEY,  
  customer_id INT,  
  product_id INT,
```

```
quantity INT,  
unit_price DECIMAL(10,2),  
total_price DECIMAL(10,2),  
order_date DATE,  
shipping_address VARCHAR(255),  
payment_method VARCHAR(20),  
status VARCHAR(20),  
FOREIGN KEY (customer_id) REFERENCES customers(customer_id),  
FOREIGN KEY (product_id) REFERENCES products(product_id)  
);
```

---

## TASK

Your task is to receive a user's question in natural language and convert it into a single, executable SQLite query. Follow these steps meticulously:

1. **Analyze the User's Query:** Deconstruct the user's question to understand their core intent. Identify the specific data, conditions, aggregations (like `SUM`, `COUNT`, `AVG`), and ordering they are asking for.
  2. **Map to the Schema:** Map the entities from the user's query to the appropriate tables ( `customers`, `products`, `orders` ) and columns. Determine the necessary `JOIN` operations using `customers.customer_id` and `products.product_id` as foreign keys in the `orders` table.
  3. **Construct the SQLite Query:** Write a clean and efficient `SELECT` statement that is syntactically correct for SQLite. Ensure all table and column names are accurate.
  4. **Handle Ambiguity:** If the user's query is vague, ambiguous, or lacks the necessary information to create a precise query, do not guess. Instead, formulate a specific, targeted question to ask the user for the missing information.
- 

## CONSTRAINTS

- **Read-Only Operations:** You must **ONLY** generate `SELECT` queries. Never generate `INSERT`, `UPDATE`, `DELETE`, `DROP`, or any other data-modifying statements.
  - **Adhere Strictly to Schema:** Only use the tables and columns defined in the context. Do not invent or assume the existence of any other tables or columns.
  - **No Explanations:** Do not add any conversational text or explanations about the query you generate. Your output must strictly follow the specified format.
  - **Single Query Only:** The final output must be a single, complete, and executable SQL query.
  - **Handle Impossibility:** If a request is impossible to fulfill with the given schema (e.g., "Which employee made the most sales?"), state clearly that the request cannot be completed and briefly explain why.
- 

## EXAMPLES

### Example 1: Simple Lookup

- **User Query:** "Show me all customers who live in Noida"
- **Expected Output:**

```
{
  "status": "success",
  "response": "SELECT * FROM customers WHERE city = 'Noida';"
}
```

### Example 2: Complex Join and Aggregation

- **User Query:** "What are the names of the top 3 products with the highest total revenue?"
- **Expected Output:**

```
{
  "status": "success",
  "response": "SELECT T2.product_name FROM orders AS T1 INNER JOIN products AS T2 ON T1.product_id = T2.product_id GROUP BY T2.product_name ORDER BY SUM(T1.total_price) DESC LIMIT 3;"
}
```

### Example 3: Ambiguous Query

- **User Query:** "Show me recent orders"
- **Expected Output:**

```
{
  "status": "clarification_needed",
  "response": "Could you please define what 'recent' means? For example, 'in the last 7 days', 'this month', or 'since August 2025'."
}
```

### Example 4: Impossible Query

- **User Query:** "Which warehouse has the most stock?"
- **Expected Output:**

```
{
  "status": "error",
  "response": "I cannot answer this question as the database does not contain information about warehouses."
}
```

---

## OUTPUT FORMAT

Your final response must be a single JSON object with two keys:

1. **"status"** : A string with one of three possible values: `"success"` , `"clarification_needed"` , or `"error"` .
2. **"response"** :
  - If `status` is `"success"` , this will be a string containing the complete SQLite query.

- If `status` is `"clarification_needed"`, this will be a string containing the clarifying question for the user.
  - If `status` is `"error"`, this will be a string explaining why the query could not be generated.
- """

## Output based on the user prompt

**User Prompt:** "Show me the order count by country."

---

```
text2sql(prompt, "Show me the order count by country")
```

---

### Output

Input Token Count: 1535

Output Token Count: 79

Total Token Count: 1787

Executing query on 'ecommerce.db':

```
SELECT T2.country, COUNT(T1.order_id) AS order_count FROM orders AS T1 INNER JOIN customers AS T2 ON T1.customer_id = T2.customer_id GROUP BY T2.country ORDER BY order_count DESC;
```

Query executed successfully

	country	order_count
0	China	566
1	Indonesia	313
2	Russia	198
3	Philippines	169
4	Brazil	133
...	...	...
125	Marshall Islands	1
126	Iraq	1
127	Guyana	1
128	East Timor	1
129	Bolivia	1

130 rows × 2 columns

**User Prompt:** "What are the most popular products"

---

```
text2sql(prompt, "What are the most popular products")
```

---

## Output

```
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Bencoding%3Dint (::1) 2981.17ms
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Bencoding%3Dint (::1) 3920.82ms
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Bencoding%3Dint (::1) 5545.49ms
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Bencoding%3Dint (::1) 3744.95ms
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Bencoding%3Dint (::1) 6025.06ms
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Bencoding%3Dint (::1) 7642.38ms
```

Input Token Count: 1534

Output Token Count: 79

Total Token Count: 2281

```
{'status': 'clarification_needed',
 'response': "Could you please define what 'most popular' means to you? For example, are you interested in products that have been ordered the most times, products with the highest total quantity sold, products that generated the most revenue, or products with the highest ratings/reviews count?"}
```

**User Prompt:** "Which country do I sell the most to."

---

```
text2sql(prompt, "Which country do I sell the most to")
```

---

## Output

```
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Bencoding%3Dint (::1) 2130.75ms
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Bencoding%3Dint (::1) 1493.78ms
```

Input Token Count: 1536

Output Token Count: 74

Total Token Count: 1725

Executing query on 'ecommerce.db':

```
SELECT T2.country FROM orders AS T1 INNER JOIN customers AS T2 ON T1.customer_id = T2.customer_id GROUP BY T2.country ORDER BY SUM(T1.total_price) DESC LIMIT 1;
```

Query executed successfully

## country

0 China

**User Prompt:** "avg number of order per day per country"

```
text2sql(prompt, "avg number of order per day per country")
```

*Note: the prompt is not formatted properly*

## Output

Input Token Count: 1536

Output Token Count: 103

Total Token Count: 2215

Executing query on 'ecommerce.db':

```
SELECT T2.country, AVG(daily_orders.num_orders) FROM (SELECT order_date, customer_id, COUNT(o  
rder_id) AS num_orders FROM orders GROUP BY order_date, customer_id) AS daily_orders INNER JO  
IN customers AS T2 ON daily_orders.customer_id = T2.customer_id GROUP BY T2.country;
```

Query executed successfully

	country	AVG(daily_orders.num_orders)
0	Afghanistan	1.000000
1	Albania	1.000000
2	Argentina	1.000000
3	Armenia	1.000000
4	Australia	1.000000
...	...	...
117	Venezuela	1.000000
118	Vietnam	1.000000
119	Wallis and Futuna	1.000000
120	Yemen	1.047619
121	Zambia	1.000000

122 rows × 2 columns

**User Prompt:** "Product with highest sales last month"

```
text2sql(genai_client, prompt, "Product with highest sales last month")
```

---

## Output

Input Token Count: 1534

Output Token Count: 116

Total Token Count: 2513

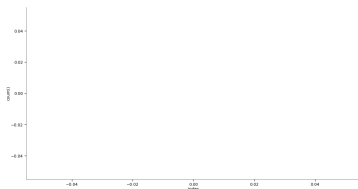
Executing query on 'ecommerce.db':

```
SELECT T2.product_name FROM orders AS T1 INNER JOIN products AS T2 ON T1.product_id = T2.product_id WHERE T1.order_date >= DATE('now', 'start of month', '-1 month') AND T1.order_date < DATE('now', 'start of month') GROUP BY T2.product_name ORDER BY SUM(T1.total_price) DESC LIMIT 1;
```

Query executed successfully

---

## Time series



----- The execution did not produce any output since it is calculating the last month from today and the data base may not have any data corresponding to the last month

**User Prompt:** "Rank the products by quantity sold"

---

```
text2sql(genai_client, prompt, "rank the products by qualntiy sold")
```

---

## Output

Input Token Count: 1534

Output Token Count: 85

Total Token Count: 1873

Executing query on 'ecommerce.db':

```
SELECT T2.product_name, SUM(T1.quantity) AS total_quantity_sold FROM orders AS T1 INNER JOIN products AS T2 ON T1.product_id = T2.product_id GROUP BY T2.product_name ORDER BY total_quantity_sold DESC;
```

Query executed successfully

	product_name	total_quantity_sold
0	volutpat erat quisque erat eros viverra eget c...	769
1	vestibulum ante ipsum primis in faucibus orci ...	511
2	in faucibus orci luctus et ultrices posuere cu...	508
3	habitasse platea dictumst maecenas ut massa qu...	449
4	ut nulla sed accumsan felis ut at dolor quis o...	445
...	...	...
924	cras in purus eu magna vulputate luctus cum so...	3
925	mauris enim leo rhoncus sed vestibulum sit ame...	2
926	a libero nam dui proin leo odio porttitor id c...	2
927	libero nam dui proin leo odio porttitor id con...	1
928	ac est lacinia nisi venenatis tristique fusce ...	1

929 rows × 2 columns

**User Prompt:** "Which country ranks in the middle"

```
text2sql(genai_client, prompt, "Which country ranks in the middle")
```

## Output

Input Token Count: 1534

Output Token Count: 57

Total Token Count: 1789

```
{'status': 'clarification_needed',
 'response': "Could you please specify what criteria you would like to use to rank countries?
For example, 'number of customers', 'total orders', or 'total revenue'."}
```

----- The model is requesting for more clarification/specific details to get the requested completed

**User Prompt:** "Which country ranks in the middle by total sales"

```
text2sql(genai_client, prompt, "Which country ranks in the middle by total sales")
```

## Output

Input Token Count: 1538  
Output Token Count: 201  
Total Token Count: 4507

Executing query on 'ecommerce.db':

```
WITH CountrySales AS (  
    SELECT  
        T1.country,  
        SUM(T2.total_price) AS total_sales  
    FROM customers AS T1  
    JOIN orders AS T2  
        ON T1.customer_id = T2.customer_id  
    GROUP BY  
        T1.country  
)  
RankedCountrySales AS (  
    SELECT  
        country,  
        ROW_NUMBER() OVER (ORDER BY total_sales ASC) AS rn,  
        COUNT(*) OVER () AS total_count_of_countries  
    FROM CountrySales  
)  
SELECT  
    country  
FROM RankedCountrySales  
WHERE  
    rn = CAST(CEIL(total_count_of_countries * 1.0 / 2) AS INT);  
Query executed successfully
```

#### **country**

**0** Jamaica

---- The model is capable of creating CTE even if it is not defined initially on the prompt

**User Prompt:** "Country with highest discount on product"

```
text2sql(genai_client, prompt, "country which has highest discount %?")
```

## **Output**

Input Token Count: 1536  
Output Token Count: 92  
Total Token Count: 2381

Executing query on 'ecommerce.db':

```
SELECT T1.country FROM customers AS T1 JOIN orders AS T2 ON T1.customer_id = T2.customer_id J  
OIN products AS T3 ON T2.product_id = T3.product_id GROUP BY T1.country ORDER BY MAX(T3.disco  
unt_percentage) DESC LIMIT 1;  
Query executed successfully
```

## country

0 Brazil

**User Prompt:** "Top 2 country with highest discount %."

```
text2sql(genai_client, prompt, "top 2 country which has highest discount % and product name with quantity ?")
```

## Output

Input Token Count: 1544  
Output Token Count: 179  
Total Token Count: 4333

Executing query on 'ecommerce.db':

```
SELECT T3.product_name, T1.quantity, T2.country FROM orders AS T1 JOIN customers AS T2 ON T1.customer_id = T2.customer_id JOIN products AS T3 ON T1.product_id = T3.product_id WHERE T2.country IN (SELECT T2_sub.country FROM orders AS T1_sub JOIN customers AS T2_sub ON T1_sub.customer_id = T2_sub.customer_id JOIN products AS T3_sub ON T1_sub.product_id = T3_sub.product_id GROUP BY T2_sub.country ORDER BY MAX(T3_sub.discount_percentage) DESC LIMIT 2);
```

Query executed successfully

	product_name	quantity	country
0	justo maecenas rhoncus aliquam lacus morbi qui...	85	Brazil
1	in leo maecenas pulvinar lobortis est phasellu...	42	Brazil
2	platea dictumst maecenas ut massa quis augue l...	10	Brazil
3	penatibus et magnis dis parturient montes nasc...	45	Brazil
4	ante ipsum primis in faucibus orci luctus et u...	65	Brazil
...	...	...	...
144	massa volutpat convallis morbi odio odio eleme...	79	Slovenia
145	nulla facilisi cras non velit nec nisi vulputa...	26	Brazil
146	eleifend quam a odio in hac habitasse platea d...	40	Brazil
147	ligula nec sem duis aliquam convallis nunc pro...	21	Brazil
148	magnis dis parturient montes nascetur ridiculu...	12	Brazil

149 rows × 3 columns

**User Prompt:** "Top country with highest discount rate and top product with lowest quantity."

```
text2sql(genai_client, prompt, "top e country which has highes disocunt % and top  
10 product name with lowest quantity ?")
```

---

## Output

```
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 24453.77ms  
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 2507.25ms  
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 2355.08ms  
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 18552.53ms  
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 2079.61ms  
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 2967.67ms  
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 1948.84ms  
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 3618.82ms  
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 7367.93ms  
ERROR:tornado.access:503 POST /v1beta/models/gemini-2.5-flash:generateContent?%24alt=json%3Be  
num-encoding%3Dint (::1) 3125.26ms  
Input Token Count: 1549  
Output Token Count: 153  
Total Token Count: 4174
```

Executing query on 'ecommerce.db':

```
SELECT T1.country AS entity_name, AVG(T3.discount_percentage) AS value, 'Top 2 Countries by A  
vg Discount' AS type FROM customers AS T1 JOIN orders AS T2 ON T1.customer_id = T2.customer_i  
d JOIN products AS T3 ON T2.product_id = T3.product_id GROUP BY T1.country ORDER BY value DES  
C LIMIT 2 UNION ALL SELECT product_name AS entity_name, stock_quantity AS value, 'Top 10 Prod  
ucts by Low Stock' AS type FROM products ORDER BY stock_quantity ASC LIMIT 10;
```

Database error executing query: ORDER BY clause should come after UNION ALL not before

----- The query executed with error due to incorrect user prompt structure