```
prompt="""### Task
Generate a SQL query to answer [QUESTION]{question}[/QUESTION]

### Instructions
-If you cannot answer the question with the available database schema,
return 'I do not know'
-Remember that revenue is price multiplied by quantity
-Remember that cost is supply_price multiplied by quantity

### Database Schema
This query will run on a database whose schema is represented in this
string:
CREATE TABLE products (
 product_id INTEGER PRIMARY KEY, --Unique ID for each product
 name VARCHAR(50), --Name of the product
 price DECIMAL (10,2), --Price of each unit of the product
 quantity INTEGER --Current quantity in stock
);

CREATE TABLE customers (
 customer_id INTEGER PRIMARY KEY, --Unique ID for each customer
 name VARCHAR(50), --Name of the customer
 address VARCHAR(100) --Mailing address of the customer
);

CREATE TABLE salespeople (
 salesperson_id INTEGER PRIMARY KEY, --Unique ID for each salesperson
 name VARCHAR(50), --Name of the salesperson
 region VARCHAR(58) --Geographic sales region
);

CREATE TABLE sales (
 sale_id INTEGER PRIMARY KEY, --Unique ID for each sale
 product_id INTEGER, --ID of product sold
 customer_id INTEGER, --ID of customer who made purchase
 salesperson_id INTEGER, --ID of salesperson who made the sale
 sale_date DATE, --Date the sale occurred
 quantity INTEGER, --Number of units sold
);
```

```
CREATE TABLE product_suppliers (
 supplier_id INTEGER PRIMARY KEY, --Unique ID for each supplier
 product_id INTEGER, --Product ID supplied
 supply_price DECIMAL(10,2) --Unit price charged by supplier
);

--sales.product_id can be joined with products.product_id
--sales.customer_id can be joined with customers.customer_id
--sales.salesperson_id can be joined with salespeople.salesperson_id
--product_suppliers.product_id can be joined with products.product_id

### Answer
Given the database schema, here is the SQL query that answers
[QUESTION]{question}[/QUESTION]
[SQL] I
"""

import sqlparse
def generate_query(question):
 updated_prompt = prompt.format(question=question)
 inputs = tokenizer(updated_prompt, return_tensors="pt").to("cuda")
 generated_ids = model.generate(
   **inputs,
   num_return_sequences=1,
   eos_token_id=tokenizer.eos_token_id,
   pad_token_id=tokenizer.eos_token_id,
   max_new_tokens = 400,
   do_sample=False,
   num_beams=1,
 )
 outputs= tokenizer.batch_decode(generated_ids, skip_special_tokens=True)

 torch.cuda.empty_cache()
 torch.cuda.synchronize()
 #empty cache so that you do generate more results w/o memory crashing
 # particularly important on Colab memory management is much more
straightforward
 #when running on an inference service
 return sqlparse.format(outputs[0].split("[SQL]") [-1], reindent=True)
```