milan@datajoin.net

http://datajoin.net

DataJoin

LLM creates SQL query using plain English request

[Document subtitle]

Milan Patel
5-2-2025

Table of Contents

Overview

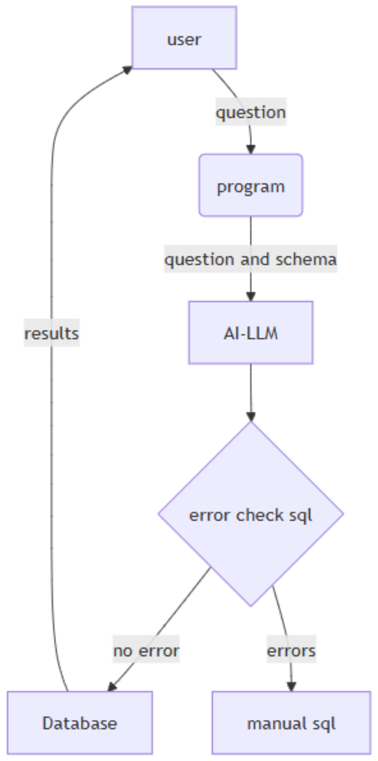
Problem: Structured data within applications contains essential operational and financial details. Accessing this information consistently necessitates IT assistance for the formulation of queries. Typically, query input formats require multiple parameters.

Solution: Users can then pose questions in plain English, which will be translated into SQL (structured query language) queries using Retrieval Augmented Generation (RAG). These queries will retrieve information from structured data within applications.

DataJoin.net provides in-depth education and consultation on integration of Large Language Models (LLM) for query development with Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) systems.
https://github.com/milan888-design/llm-creates-sql

Fundamentals of LLM to generate SQL
In SQL schema, data is stored in multiple tables, a table has multiple columns. Typically, a table represents, and entity and columns are attributes of an entity. Since in real world, entities have relationships with other entities, table columns have relationships to other tables using foreign keys. It is possible that the names of tables and their columns are “normal” English words, however, in some cases the names can be abbreviation or not truly representing an entity. End user wants to ask a question in plain English; however, a programmer must translate English into SQL. This task is done by LLM. This is not 100% accurate at this point.

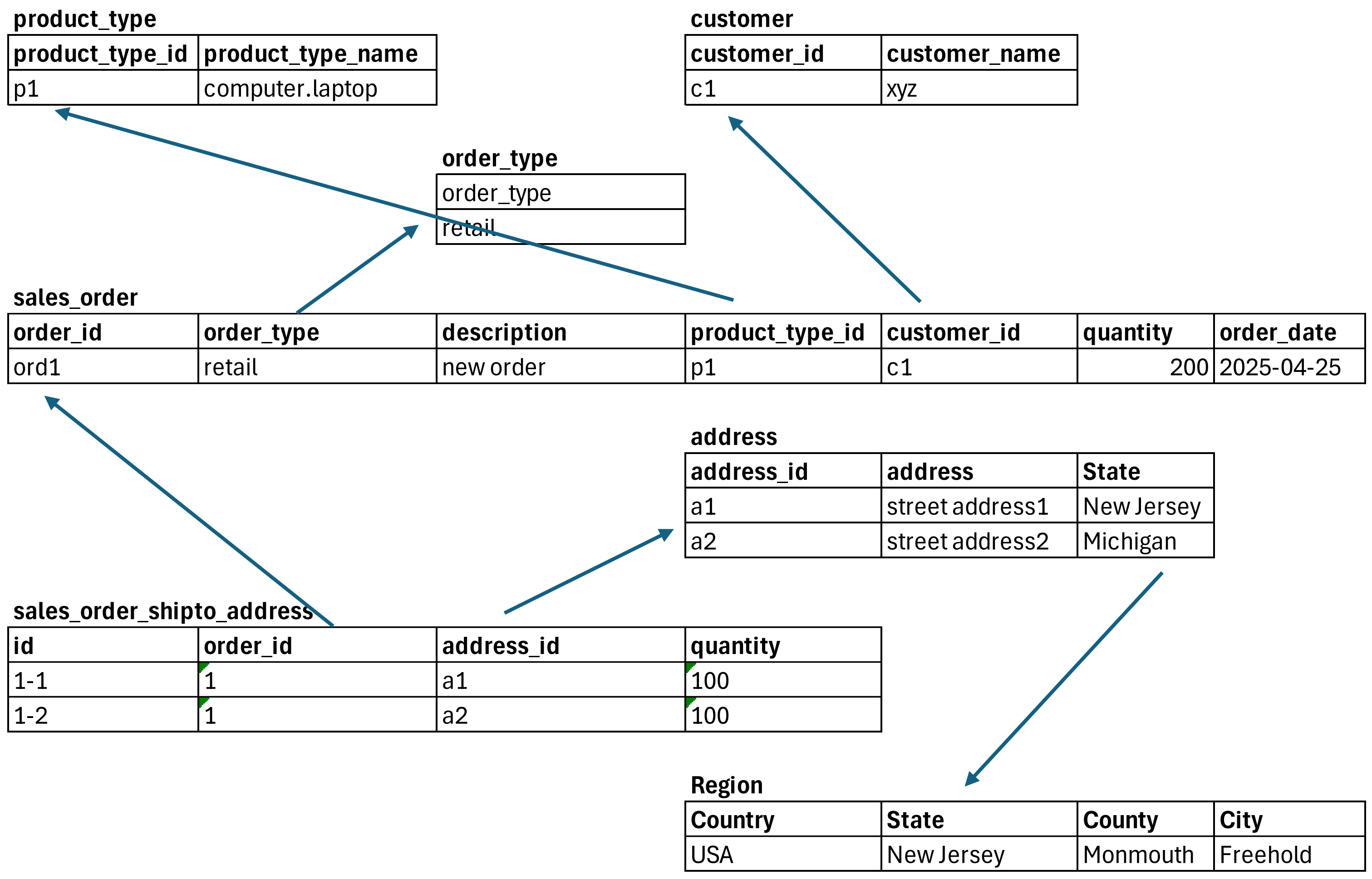
Flowchart


user
question
program
question and schema
AI-LLM
error check sql
no error
errors
Database
manual sql
results

Schema types

Production database with data (table names in plain English and Foreign keys defined)

This is an example of a SQL schema.



product_type

product_type_id	product_type_name
p1	computer.laptop

customer

customer_id	customer_name
c1	xyz

order_type

order_type
retail

sales_order

order_id	order_type	description	product_type_id	customer_id	quantity	order_date
ord1	retail	new order	p1	c1	200	2025-04-25

address

address_id	address	State
a1	street address1	New Jersey
a2	street address2	Michigan

sales_order_shipto_address

id	order_id	address_id	quantity
1-1	1	a1	100
1-2	1	a2	100

Region

Country	State	County	City
USA	New Jersey	Monmouth	Freehold

note1: customer and sales order tables are item(instance) tables
note2: product_type and order_type are category (class/type) tables
note3: sales_order_shipto_address table is many to many relations between sales_order and address tables
note4: arrows indicate foreign keys are defined in table definition

Production database with data (table names not in plain English and or foreign keys not defined)

Create views using plain English in the same production db.

ptype

product_type_id	product_type_name
p1	computer.laptop

create view product_type as...

cc

customer_id	customer_name
c1	xyz

create view customer as...

ot

order_type
salesorder.retail

create view order_type as...

so

order_id	order_type	description	product_type_id	customer_id	quantity	order_date
ord1	retail	new order	p1	c1	200	2025-04-25

create view sales_order as...

ad

address_id	address	State
a1	street address1	New Jersey
a2	street address2	Michigan

create view address as...

so_ad

id	order_id	address_id	quantity
1-1	1	a1	100
1-2	1	a2	100

create view sales_order_shipto_address as...

rg

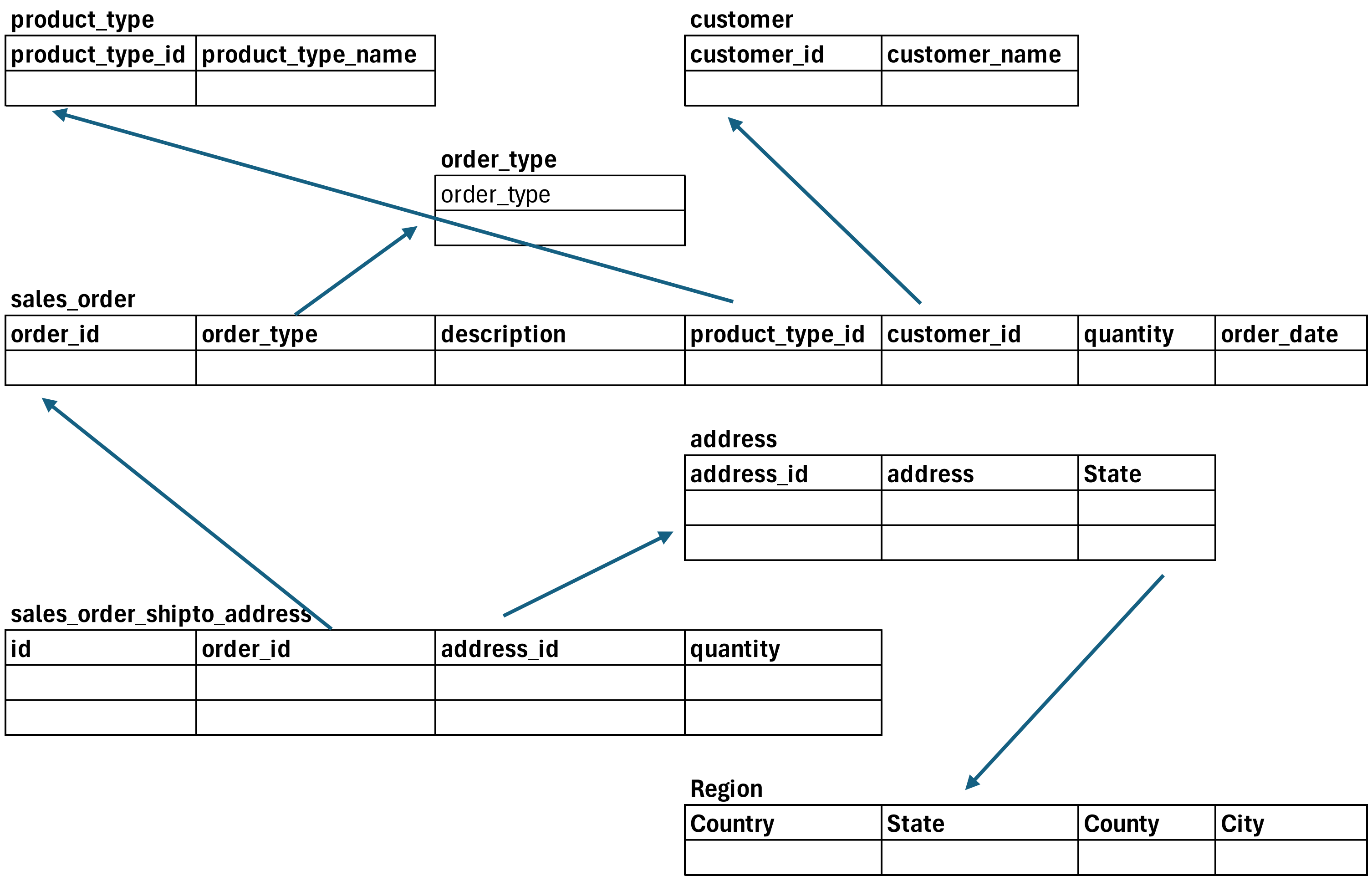
Country	State	County	City
USA	New Jersey	Monmouth	Freehold

note1: customer and sales order tables are item(instance) tables
note2: product_type and order_type are category (class/type) tables
note3: sales_order_shipto_address table is many to many relations between sales_order and address tables

Schema changes

Parallel database with no data (table definition with foreign keys)

This database has no data. It is used to generate SQL queries only. Then, SQL query is run on actual production db. This schema table should be in plain English. This is needed when the actual production db tables are not in plain English.



product_type

product_type_id	product_type_name

customer

customer_id	customer_name

order_type

order_type

sales_order

order_id	order_type	description	product_type_id	customer_id	quantity	order_date

address

address_id	address	State

sales_order_shipto_address

id	order_id	address_id	quantity

Region

Country	State	County	City

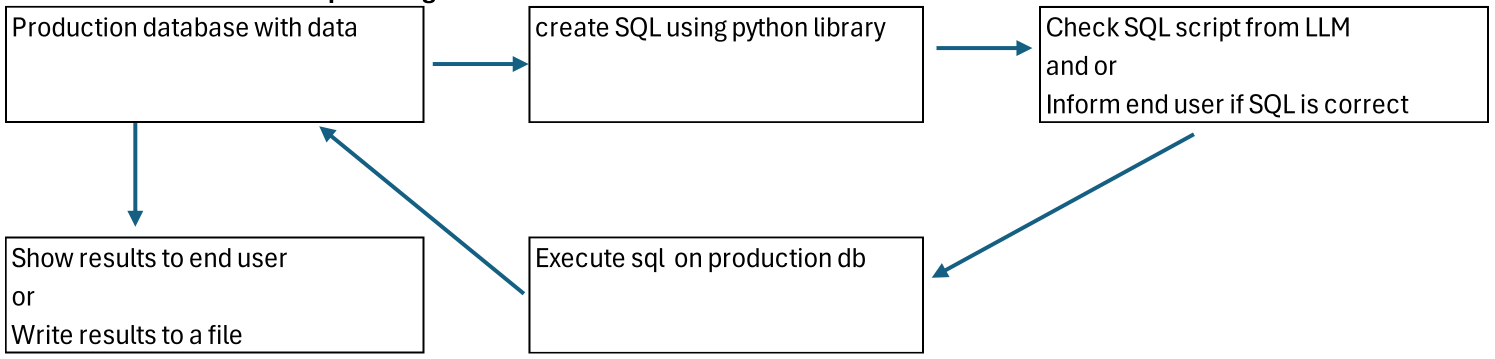
note1: customer and sales order tables are item(instance) tables
note2: product_type and order_type are category (class/type) tables
note3: sales_order_shipto_address table is many to many relations between sales_order and address tables

note4: arrows indicate foreign keys are defined in table definition

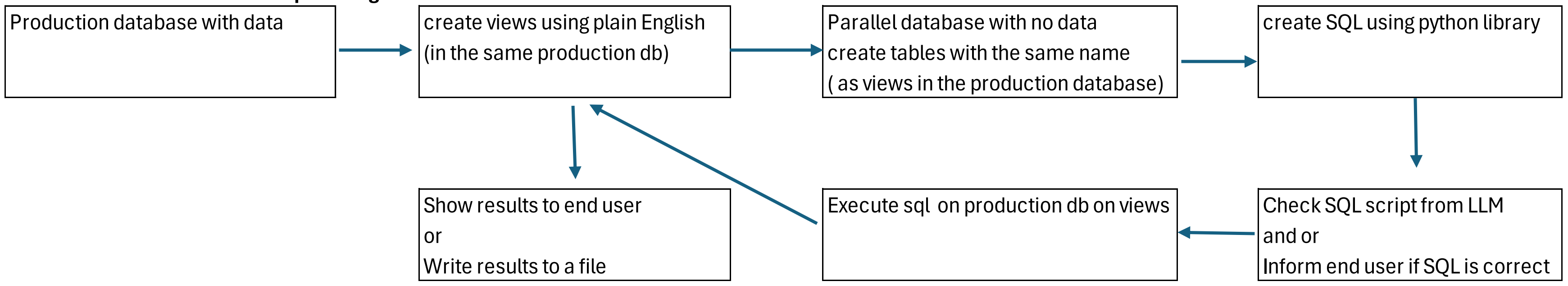
Decision tree for a schema

The design of the current SQL Schema influences what needs to be done for LLM assisted SQL queries.

Scenario 1: table names in plain English


Production database with data
create SQL using python library
Check SQL script from LLM and or Inform end user if SQL is correct
Execute sql on production db
Show results to end user or Write results to a file

Scenario 2: table names not in plain English


Production database with data
create views using plain English (in the same production db)
Parallel database with no data create tables with the same name (as views in the production database)
create SQL using python library
Check SQL script from LLM and or Inform end user if SQL is correct
Execute sql on production db on views
Show results to end user or Write results to a file

How LLM generate SQL

LLM needs to know the SQL schema to generate SQL to answer your question

LLM uses SQL definition of foreign key to understand the relationship between tables.

LLM can work without the foreign key definition as long as it finds common attribute names. For example, if year column is named as year or year_name across all tables, then, LLM knows how to use it to join tables.

PostgreSQL database and Python code

Create chinook database using admin UI. Run chinook.sql to create tables and data in the tables.

Modify the question and dbconnection as per your situation in LLM_creates_sql.py. Then, run LLM_creates_sql.py

```python
from openai import OpenAI
from langchain_openai import ChatOpenAI
from langchain.chains import create_sql_query_chain
from langchain_community.utilities import SQLDatabase
OpenAI.api_key = "YOUR KEY"
# Initialize the OpenAI client
client = OpenAI()
dbconnection='postgresql://postgres:pass@localhost:5432/chinook'
question1 = 'List invoices and customer names for customers living in Brazil'
db = SQLDatabase.from_uri(dbconnection)
llm = ChatOpenAI(model='gpt-3.5-turbo', temperature=0)
chain = create_sql_query_chain(llm, db)
response = chain.invoke({"question": question1})
result1=db.run(response)
sqlandresult=response +"---"+result1
```

SELECT i.invoice_id, c.first_name, c.last_name
FROM invoice i

```
JOIN customer c ON i.customer_id = c.customer_id
WHERE c.country = 'Brazil'
LIMIT 5;---[(25, 'Eduardo', 'Martins'), (34, 'Roberto', 'Almeida'), (35, 'Fernanda', 'Ramos'), (57, 'Alexandre', 'Rocha'), (58, 'Fernanda', 'Ramos')]
```

Example questions / prompts that work with chat GPT

```
--question1 = "List invoices and customer names for customers living in Brazil"
SELECT i.invoice_id, c.first_name, c.last_name
FROM invoice i
JOIN customer c ON i.customer_id = c.customer_id
WHERE c.country = 'Brazil'
LIMIT 5;

--question1 = "List invoice number, amount and customer names for customers living in Brazil"
SELECT i.invoice_id, i.total, c.first_name, c.last_name
FROM invoice i
JOIN customer c ON i.customer_id = c.customer_id
WHERE c.country = 'Brazil'
LIMIT 5;

--question1 = "List invoice number, amount and customer names for customers living in Brazil"
SELECT i.invoice_id,c.customer_id, i.total, c.first_name, c.last_name
FROM invoice i,customer c
 where i.customer_id = c.customer_id
 and  c.country = 'Brazil'
LIMIT 5;

--question1 = "Show all invoices along with the customer details."
SELECT i.invoice_id, i.invoice_date, i.total, c.first_name, c.last_name, c.email
FROM invoice i
JOIN customer c ON i.customer_id = c.customer_id
ORDER BY i.invoice_date DESC
```

```
LIMIT 5;

--question1 = "Retrieve all employees and their respective managers."
SELECT e1.first_name AS employee_first_name, e1.last_name AS employee_last_name, e2.first_name AS manager_first_name,
e2.last_name AS manager_last_name
FROM employee e1
LEFT JOIN employee e2 ON e1.reports_to = e2.employee_id
ORDER BY e1.employee_id
LIMIT 5;

--question1 = "List all tracks along with their album names and artist names."
SELECT
  "t"."name" AS "track_name",
 "al"."title" AS "album_name",
  "ar"."name" AS "artist_name"
FROM
  "track" AS "t"
JOIN "album" AS "al" ON "t"."album_id" = "al"."album_id"
JOIN "artist" AS "ar" ON "al"."artist_id" = "ar"."artist_id"
LIMIT 5;

--the following came from chatgpt and it made a mistake. There is no name for the album. It is title
--also there is no able a with name. It is al with title
SELECT
  "t"."name" AS "track_name",
  "a"."name" AS "album_name",
  "ar"."name" AS "artist_name"
FROM
  "track" AS "t"
JOIN "album" AS "al" ON "t"."album_id" = "al"."album_id"
JOIN "artist" AS "ar" ON "al"."artist_id" = "ar"."artist_id"
LIMIT 5;
```

```
--question1 = "List all tracks along with their album names and artist names and customer names who bought it."
--the following joins 5 tables
SELECT
   "t"."name" AS "track_name",
   "al"."title" AS "album_title",
   "ar"."name" AS "artist_name",
   "c"."first_name" AS "customer_first_name",
   "c"."last_name" AS "customer_last_name"
FROM
   "track" "t"
JOIN "album" "al" ON "t"."album_id" = "al"."album_id"
JOIN "artist" "ar" ON "al"."artist_id" = "ar"."artist_id"
JOIN "invoice_line" "il" ON "t"."track_id" = "il"."track_id"
JOIN "invoice" "i" ON "il"."invoice_id" = "i"."invoice_id"
JOIN "customer" "c" ON "i"."customer_id" = "c"."customer_id"
ORDER BY "t"."name"
LIMIT 5;
```