SQL

This example demonstrates the use of the `SQLDatabaseChain` for answering questions over a SQL database.

Under the hood, LangChain uses SQLAlchemy to connect to SQL databases. The `SQLDatabaseChain` can therefore be used with any SQL dialect supported by SQLAlchemy, such as MS SQL, MySQL, MariaDB, PostgreSQL, Oracle SQL, Databricks and SQLite. Please refer to the SQLAlchemy documentation for more information about requirements for connecting to your database. For example, a connection to MySQL requires an appropriate connector such as PyMySQL. A URI for a MySQL connection might look like: `mysql+pymysql://user:pass@some_mysql_db_address/db_name`.

This demonstration uses SQLite and the example Chinook database. To set it up, follow the instructions on https://database.guide/2-sample-databases-sqlite/, placing the `.db` file in a notebooks folder at the root of this repository.

```
from langchain import OpenAI, SQLDatabase, SQLDatabaseChain
```

```
db = SQLDatabase.from_uri("sqlite:///../../../../notebooks/Chinook.db")
llm = OpenAI(temperature=0, verbose=True)
```

NOTE: For data-sensitive projects, you can specify `return_direct=True` in the `SQLDatabaseChain` initialization to directly return the output of the SQL query without any additional formatting. This prevents the LLM from seeing any contents within the database. Note, however, the LLM still has access to the database scheme (i.e. dialect, table and key names) by default.

```
db_chain = SQLDatabaseChain.from_llm(llm, db, verbose=True)
```

```
db_chain.run("How many employees are there?")
```

```
    > Entering new SQLDatabaseChain chain...
    How many employees are there?
    SQLQuery:

    /workspace/langchain/langchain/sql_database.py:191: SAWarning: Dialect sqlite+pysqlite does *not* support
Decimal objects natively, and SQLAlchemy must convert from floating point - rounding errors and other issues
may occur. Please consider storing Decimal numbers as strings or integers on this platform for lossless
storage.
      sample_rows = connection.execute(command)


    SELECT COUNT(*) FROM "Employee";
    SQLResult: [(8,)]
    Answer:There are 8 employees.
    > Finished chain.



    'There are 8 employees.'
```

Use Query Checker

Sometimes the Language Model generates invalid SQL with small mistakes that can be self-corrected using the same technique used by the SQL Database Agent to try and fix the SQL using the LLM. You can simply specify this option when creating the chain:

```
db_chain = SQLDatabaseChain.from_llm(llm, db, verbose=True, use_query_checker=True)
```

```
db_chain.run("How many albums by Aerosmith?")
```

```


    > Entering new SQLDatabaseChain chain...
    How many albums by Aerosmith?
    SQLQuery:SELECT COUNT(*) FROM Album WHERE ArtistId = 3;
    SQLResult: [(1,)]
    Answer:There is 1 album by Aerosmith.
    > Finished chain.



    'There is 1 album by Aerosmith.'
```

Customize Prompt

You can also customize the prompt that is used. Here is an example prompting it to understand that foobar is the same as the Employee table

```
from langchain.prompts.prompt import PromptTemplate

_DEFAULT_TEMPLATE = """Given an input question, first create a syntactically correct {dialect} query to run,
then look at the results of the query and return the answer.
Use the following format:

Question: "Question here"
SQLQuery: "SQL Query to run"
SQLResult: "Result of the SQLQuery"
Answer: "Final answer here"

Only use the following tables:

{table_info}

If someone asks for the table foobar, they really mean the employee table.

Question: {input}"""
PROMPT = PromptTemplate(
    input_variables=["input", "table_info", "dialect"], template=_DEFAULT_TEMPLATE
)
```

```
db_chain = SQLDatabaseChain.from_llm(llm, db, prompt=PROMPT, verbose=True)
```

```
db_chain.run("How many employees are there in the foobar table?")
```

```
> Entering new SQLDatabaseChain chain...
How many employees are there in the foobar table?
SQLQuery:SELECT COUNT(*) FROM Employee;
SQLResult: [(8,)]
Answer:There are 8 employees in the foobar table.
> Finished chain.
```

```
'There are 8 employees in the foobar table.'
```

Return Intermediate Steps

You can also return the intermediate steps of the SQLDatabaseChain. This allows you to access the SQL statement that was generated, as well as the result of running that against the SQL Database.

```
db_chain = SQLDatabaseChain.from_llm(llm, db, prompt=PROMPT, verbose=True, use_query_checker=True,
return_intermediate_steps=True)
```

```
result = db_chain("How many employees are there in the foobar table?")
result["intermediate_steps"]
```

```
> Entering new SQLDatabaseChain chain...
How many employees are there in the foobar table?
SQLQuery:SELECT COUNT(*) FROM Employee;
SQLResult: [(8,)]
Answer:There are 8 employees in the foobar table.
> Finished chain.
```

```
[{'input': 'How many employees are there in the foobar table?\nSQLQuery:SELECT COUNT(*) FROM Employee;\nSQLF
  'top_k': '5',
  'dialect': 'sqlite',
  'table_info': '\nCREATE TABLE "Artist" (\n\t"ArtistId" INTEGER NOT NULL, \n\t"Name" NVARCHAR(120), \n\tPRI
table:\nArtistId\tName\n1\tAC/DC\n2\tAccept\n3\tAerosmith\n*/\n\n\nCREATE TABLE "Employee" (\n\t"EmployeeId" INT
\n\t"FirstName" NVARCHAR(20) NOT NULL, \n\t"Title" NVARCHAR(30), \n\t"ReportsTo" INTEGER, \n\t"BirthDate" DATETI
NVARCHAR(70), \n\t"City" NVARCHAR(40), \n\t"State" NVARCHAR(40), \n\t"Country" NVARCHAR(40), \n\t"PostalCode" NV
NVARCHAR(24), \n\t"Email" NVARCHAR(60), \n\tPRIMARY KEY ("EmployeeId"), \n\tFOREIGN KEY("ReportsTo") REFERENCES
Employee
table:\nEmployeeId\tLastName\tFirstName\tTitle\tReportsTo\tBirthDate\tHireDate\tAddress\tCity\tState\tCountry\tP
l Manager\tNone\t1962-02-18 00:00:00\t2002-08-14 00:00:00\t11120 Jasper Ave NW\tEdmonton\tAB\tCanada\tT5K 2N1\t+
3457\tandrew@chinookcorp.com\n2\tEdwards\tNancy\tSales Manager\t1\t1958-12-08 00:00:00\t2002-05-01 00:00:00\t825
3443\t+1 (403) 262-3322\tnancy@chinookcorp.com\n3\tPeacock\tJane\tSales Support Agent\t2\t1973-08-29 00:00:00\t2
SW\tCalgary\tAB\tCanada\tT2P 5M5\t+1 (403) 262-3443\t+1 (403) 262-6712\tjane@chinookcorp.com\n*/\n\n\nCREATE TAB
NVARCHAR(120), \n\tPRIMARY KEY ("GenreId")\n)\n\n/*\n3 rows from Genre table:\nGenreId\tName\n1\tRock\n2\tJazz\r
(\n\t"MediaTypeId" INTEGER NOT NULL, \n\t"Name" NVARCHAR(120), \n\tPRIMARY KEY ("MediaTypeId")\n)\n\n/*\n3 rows
audio file\n2\tProtected AAC audio file\n3\tProtected MPEG-4 video file\n*/\n\n\nCREATE TABLE "Playlist" (\n\t"P
\n\tPRIMARY KEY ("PlaylistId")\n)\n\n/*\n3 rows from Playlist table:\nPlaylistId\tName\n1\tMusic\n2\tMovies\n3\t
INTEGER NOT NULL, \n\t"Title" NVARCHAR(160) NOT NULL, \n\t"ArtistId" INTEGER NOT NULL, \n\tPRIMARY KEY ("AlbumId
```

```
("ArtistId")\n)\n\n/*\n3 rows from Album table:\nAlbumId\tTitle\tArtistId\n1\tFor Those About To Rock We Salute
Wild\t2\n*/\n\n\nCREATE TABLE "Customer" (\n\t"CustomerId" INTEGER NOT NULL, \n\t"FirstName" NVARCHAR(40) NOT NU
\n\t"Company" NVARCHAR(80), \n\t"Address" NVARCHAR(70), \n\t"City" NVARCHAR(40), \n\t"State" NVARCHAR(40), \n\t'
\n\t"Phone" NVARCHAR(24), \n\t"Fax" NVARCHAR(24), \n\t"Email" NVARCHAR(60) NOT NULL, \n\t"SupportRepId" INTEGER,
KEY("SupportRepId") REFERENCES "Employee" ("EmployeeId")\n)\n\n/*\n3 rows from Customer
table:\nCustomerId\tFirstName\tLastName\tCompany\tAddress\tCity\tState\tCountry\tPostalCode\tPhone\tFax\tEmail\t
Brasileira de Aeronáutica S.A.\tAv. Brigadeiro Faria Lima, 2170\tSão José dos Campos\tSP\tBrazil\t12227-000\t+55
5566\tluisg@embraer.com.br\t3\n2\tLeonie\tKöhler\tNone\tTheodor-Heuss-Straße 34\tStuttgart\tNone\tGermany\t70174
2842222\tNone\tleonekohler@surfeu.de\t5\n3\tFrançois\tTremblay\tNone\t1498 rue Bélanger\tMontréal\tQC\tCanada\tH
4711\tNone\tftremblay@gmail.com\t3\n*/\n\n\nCREATE TABLE "Invoice" (\n\t"InvoiceId" INTEGER NOT NULL, \n\t"Custo
NOT NULL, \n\t"BillingAddress" NVARCHAR(70), \n\t"BillingCity" NVARCHAR(40), \n\t"BillingState" NVARCHAR(40), \r
\n\t"BillingPostalCode" NVARCHAR(10), \n\t"Total" NUMERIC(10, 2) NOT NULL, \n\tPRIMARY KEY ("InvoiceId"), \n\tFO
("CustomerId")\n)\n\n/*\n3 rows from Invoice
table:\nInvoiceId\tCustomerId\tInvoiceDate\tBillingAddress\tBillingCity\tBillingState\tBillingCountry\tBillingPo
Heuss-Straße 34\tStuttgart\tNone\tGermany\t70174\t1.98\n2\t4\t2009-01-02 00:00:00\tUllevålsveien 14\tOslo\tNone\
00:00:00\tGrétrystraat 63\tBrussels\tNone\tBelgium\t1000\t5.94\n*/\n\n\nCREATE TABLE "Track" (\n\t"TrackId" INTE
\n\t"AlbumId" INTEGER, \n\t"MediaTypeId" INTEGER NOT NULL, \n\t"GenreId" INTEGER, \n\t"Composer" NVARCHAR(220),
INTEGER, \n\t"UnitPrice" NUMERIC(10, 2) NOT NULL, \n\tPRIMARY KEY ("TrackId"), \n\tFOREIGN KEY("MediaTypeId") RE
KEY("GenreId") REFERENCES "Genre" ("GenreId"), \n\tFOREIGN KEY("AlbumId") REFERENCES "Album" ("AlbumId")\n)\n\n/
table:\nTrackId\tName\tAlbumId\tMediaTypeId\tGenreId\tComposer\tMilliseconds\tBytes\tUnitPrice\n1\tFor Those Abo
Malcolm Young, Brian Johnson\t343719\t11170334\t0.99\n2\tBalls to the Wall\t2\t2\t1\tNone\t342562\t5510424\t0.99
U. Dirkscneider & W. Hoffman\t230619\t3990994\t0.99\n*/\n\n\nCREATE TABLE "InvoiceLine" (\n\t"InvoiceLineId" INT
\n\t"TrackId" INTEGER NOT NULL, \n\t"UnitPrice" NUMERIC(10, 2) NOT NULL, \n\t"Quantity" INTEGER NOT NULL, \n\tPF
KEY("TrackId") REFERENCES "Track" ("TrackId"), \n\tFOREIGN KEY("InvoiceId") REFERENCES "Invoice" ("InvoiceId")\r
table:\nInvoiceLineId\tInvoiceId\tTrackId\tUnitPrice\tQuantity\n1\t1\t2\t0.99\t1\n2\t1\t4\t0.99\t1\n3\t2\t6\t0.9
(\n\t"PlaylistId" INTEGER NOT NULL, \n\t"TrackId" INTEGER NOT NULL, \n\tPRIMARY KEY ("PlaylistId", "TrackId"), \
("TrackId"), \n\tFOREIGN KEY("PlaylistId") REFERENCES "Playlist" ("PlaylistId")\n)\n\n/*\n3 rows from PlaylistTr
table:\nPlaylistId\tTrackId\n1\t3402\n1\t3389\n1\t3390\n*/',
      'stop': ['\nSQLResult:']},
     'SELECT COUNT(*) FROM Employee;',
     {'query': 'SELECT COUNT(*) FROM Employee;', 'dialect': 'sqlite'},
```

```
    'SELECT COUNT(*) FROM Employee;',
    '[(8,)]']
```

Choosing how to limit the number of rows returned

If you are querying for several rows of a table you can select the maximum number of results you want to get by using the 'top_k' parameter (default is 10). This is useful for avoiding query results that exceed the prompt max length or consume tokens unnecessarily.

```
db_chain = SQLDatabaseChain.from_llm(llm, db, verbose=True, use_query_checker=True, top_k=3)
```

```
db_chain.run("What are some example tracks by composer Johann Sebastian Bach?")
```

```
    > Entering new SQLDatabaseChain chain...
    What are some example tracks by composer Johann Sebastian Bach?
    SQLQuery:SELECT Name FROM Track WHERE Composer = 'Johann Sebastian Bach' LIMIT 3
    SQLResult: [('Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace',), ('Aria Mit 30 Veränderungen, BWV
988 "Goldberg Variations": Aria',), ('Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude',)]
    Answer:Examples of tracks by Johann Sebastian Bach are Concerto for 2 Violins in D Minor, BWV 1043: I.
Vivace, Aria Mit 30 Veränderungen, BWV 988 "Goldberg Variations": Aria, and Suite for Solo Cello No. 1 in G
Major, BWV 1007: I. Prélude.
    > Finished chain.
```

```
    'Examples of tracks by Johann Sebastian Bach are Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace, Aria Mit 30 Veränderungen, BWV 988 "Goldberg Variations": Aria, and Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude.'
```

Adding example rows from each table

Sometimes, the format of the data is not obvious and it is optimal to include a sample of rows from the tables in the prompt to allow the LLM to understand the data before providing a final query. Here we will use this feature to let the LLM know that artists are saved with their full names by providing two rows from the `Track` table.

```
db = SQLDatabase.from_uri(
    "sqlite:///../../../../notebooks/Chinook.db",
    include_tables=['Track'], # we include only one table to save tokens in the prompt :)
    sample_rows_in_table_info=2)
```

The sample rows are added to the prompt after each corresponding table's column information:

```
print(db.table_info)
```

```
    CREATE TABLE "Track" (
        "TrackId" INTEGER NOT NULL,
        "Name" NVARCHAR(200) NOT NULL,
```

```
        "AlbumId" INTEGER,
        "MediaTypeId" INTEGER NOT NULL,
        "GenreId" INTEGER,
        "Composer" NVARCHAR(220),
        "Milliseconds" INTEGER NOT NULL,
        "Bytes" INTEGER,
        "UnitPrice" NUMERIC(10, 2) NOT NULL,
        PRIMARY KEY ("TrackId"),
        FOREIGN KEY("MediaTypeId") REFERENCES "MediaType" ("MediaTypeId"),
        FOREIGN KEY("GenreId") REFERENCES "Genre" ("GenreId"),
        FOREIGN KEY("AlbumId") REFERENCES "Album" ("AlbumId")
    )

    /*
    2 rows from Track table:
    TrackId Name    AlbumId MediaTypeId GenreId Composer    Milliseconds    Bytes   UnitPrice
    1   For Those About To Rock (We Salute You) 1   1   1   Angus Young, Malcolm Young, Brian Johnson
343719  11170334    0.99
    2   Balls to the Wall   2   2   1   None    342562  5510424 0.99
    */
```

```
db_chain = SQLDatabaseChain.from_llm(llm, db, use_query_checker=True, verbose=True)
```

```
db_chain.run("What are some example tracks by Bach?")
```

```


    > Entering new SQLDatabaseChain chain...
    What are some example tracks by Bach?
    SQLQuery:SELECT "Name", "Composer" FROM "Track" WHERE "Composer" LIKE '%Bach%' LIMIT 5
```

```
    SQLResult: [('American Woman', 'B. Cummings/G. Peterson/M.J. Kale/R. Bachman'), ('Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace', 'Johann Sebastian Bach'), ('Aria Mit 30 Veränderungen, BWV 988 "Goldberg Variations": Aria', 'Johann Sebastian Bach'), ('Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude', 'Johann Sebastian Bach'), ('Toccata and Fugue in D Minor, BWV 565: I. Toccata', 'Johann Sebastian Bach')]
    Answer:Tracks by Bach include 'American Woman', 'Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace', 'Aria Mit 30 Veränderungen, BWV 988 "Goldberg Variations": Aria', 'Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude', and 'Toccata and Fugue in D Minor, BWV 565: I. Toccata'.
    > Finished chain.
```

```
    'Tracks by Bach include \'American Woman\', \'Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace\', \'Aria Mit 30 Veränderungen, BWV 988 "Goldberg Variations": Aria\', \'Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude\', and \'Toccata and Fugue in D Minor, BWV 565: I. Toccata\'.'
```

Custom Table Info

In some cases, it can be useful to provide custom table information instead of using the automatically generated table definitions and the first `sample_rows_in_table_info` sample rows. For example, if you know that the first few rows of a table are uninformative, it could help to manually provide example rows that are more diverse or provide more information to the model. It is also possible to limit the columns that will be visible to the model if there are unnecessary columns.

This information can be provided as a dictionary with table names as the keys and table information as the values. For example, let's provide a custom definition and sample rows for the Track table with only a few columns:

```
custom_table_info = {
    "Track": """CREATE TABLE Track (
    "TrackId" INTEGER NOT NULL,
    "Name" NVARCHAR(200) NOT NULL,
```

```
    "Composer" NVARCHAR(220),
    PRIMARY KEY ("TrackId")
)
/*
3 rows from Track table:
TrackId Name    Composer
1   For Those About To Rock (We Salute You) Angus Young, Malcolm Young, Brian Johnson
2   Balls to the Wall   None
3   My favorite song ever   The coolest composer of all time
*/"""
}
```

```
db = SQLDatabase.from_uri(
    "sqlite:///../../../../notebooks/Chinook.db",
    include_tables=['Track', 'Playlist'],
    sample_rows_in_table_info=2,
    custom_table_info=custom_table_info)

print(db.table_info)
```

```

    CREATE TABLE "Playlist" (
        "PlaylistId" INTEGER NOT NULL,
        "Name" NVARCHAR(120),
        PRIMARY KEY ("PlaylistId")
    )

    /*
    2 rows from Playlist table:
    PlaylistId  Name
```

```
    1   Music
    2   Movies
    */

    CREATE TABLE Track (
        "TrackId" INTEGER NOT NULL,
        "Name" NVARCHAR(200) NOT NULL,
        "Composer" NVARCHAR(220),
        PRIMARY KEY ("TrackId")
    )
    /*
    3 rows from Track table:
    TrackId Name    Composer
    1   For Those About To Rock (We Salute You) Angus Young, Malcolm Young, Brian Johnson
    2   Balls to the Wall   None
    3   My favorite song ever   The coolest composer of all time
    */
```

Note how our custom table definition and sample rows for `Track` overrides the `sample_rows_in_table_info` parameter. Tables that are not overridden by `custom_table_info`, in this example `Playlist`, will have their table info gathered automatically as usual.

```
db_chain = SQLDatabaseChain.from_llm(llm, db, verbose=True)
db_chain.run("What are some example tracks by Bach?")
```

```
    > Entering new SQLDatabaseChain chain...
    What are some example tracks by Bach?
    SQLQuery:SELECT "Name" FROM Track WHERE "Composer" LIKE '%Bach%' LIMIT 5;
    SQLResult: [('American Woman',), ('Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace',), ('Aria Mit
30 Veränderungen, BWV 988 "Goldberg Variations": Aria',), ('Suite for Solo Cello No. 1 in G Major, BWV 1007:
```

```
I. Prélude',), ('Toccata and Fugue in D Minor, BWV 565: I. Toccata',)]
    Answer:text='You are a SQLite expert. Given an input question, first create a syntactically correct SQLite query to run, then look at the results of the query and return the answer to the input question.\nUnless the user specifies in the question a specific number of examples to obtain, query for at most 5 results using the LIMIT clause as per SQLite. You can order the results to return the most informative data in the database.\nNever query for all columns from a table. You must query only the columns that are needed to answer the question. Wrap each column name in double quotes (") to denote them as delimited identifiers.\nPay attention to use only the column names you can see in the tables below. Be careful to not query for columns that do not exist. Also, pay attention to which column is in which table.\n\nUse the following format:\n\nQuestion: "Question here"\nSQLQuery: "SQL Query to run"\nSQLResult: "Result of the SQLQuery"\nAnswer: "Final answer here"\n\nOnly use the following tables:\n\nCREATE TABLE "Playlist" (\n\t"PlaylistId" INTEGER NOT NULL, \n\t"Name" NVARCHAR(120), \n\tPRIMARY KEY ("PlaylistId")\n)\n\n/*\n2 rows from Playlist table:\nPlaylistId\tName\n1\tMusic\n2\tMovies\n*/\n\nCREATE TABLE Track (\n\t"TrackId" INTEGER NOT NULL, \n\t"Name" NVARCHAR(200) NOT NULL,\n\t"Composer" NVARCHAR(220),\n\tPRIMARY KEY ("TrackId")\n)\n/*\n3 rows from Track table:\nTrackId\tName\tComposer\n1\tFor Those About To Rock (We Salute You)\tAngus Young, Malcolm Young, Brian Johnson\n2\tBalls to the Wall\tNone\n3\tMy favorite song ever\tThe coolest composer of all time\n*/\n\nQuestion: What are some example tracks by Bach?\nSQLQuery:SELECT "Name" FROM Track WHERE "Composer" LIKE \'%Bach%\' LIMIT 5;\nSQLResult: [(\'American Woman\',), (\'Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace\',), (\'Aria Mit 30 Veränderungen, BWV 988 "Goldberg Variations": Aria\',), (\'Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude\',), (\'Toccata and Fugue in D Minor, BWV 565: I. Toccata\',)]\nAnswer:'
    You are a SQLite expert. Given an input question, first create a syntactically correct SQLite query to run, then look at the results of the query and return the answer to the input question.
    Unless the user specifies in the question a specific number of examples to obtain, query for at most 5 results using the LIMIT clause as per SQLite. You can order the results to return the most informative data in the database.
    Never query for all columns from a table. You must query only the columns that are needed to answer the question. Wrap each column name in double quotes (") to denote them as delimited identifiers.
    Pay attention to use only the column names you can see in the tables below. Be careful to not query for columns that do not exist. Also, pay attention to which column is in which table.

    Use the following format:
```

```
Question: "Question here"
SQLQuery: "SQL Query to run"
SQLResult: "Result of the SQLQuery"
Answer: "Final answer here"

Only use the following tables:

CREATE TABLE "Playlist" (
    "PlaylistId" INTEGER NOT NULL,
    "Name" NVARCHAR(120),
    PRIMARY KEY ("PlaylistId")
)

/*
2 rows from Playlist table:
PlaylistId  Name
1   Music
2   Movies
*/

CREATE TABLE Track (
    "TrackId" INTEGER NOT NULL,
    "Name" NVARCHAR(200) NOT NULL,
    "Composer" NVARCHAR(220),
    PRIMARY KEY ("TrackId")
)
/*
3 rows from Track table:
TrackId Name    Composer
1   For Those About To Rock (We Salute You) Angus Young, Malcolm Young, Brian Johnson
2   Balls to the Wall   None
3   My favorite song ever   The coolest composer of all time
*/
```

```
    Question: What are some example tracks by Bach?
    SQLQuery:SELECT "Name" FROM Track WHERE "Composer" LIKE '%Bach%' LIMIT 5;
    SQLResult: [('American Woman',), ('Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace',), ('Aria Mit 30 Veränderungen, BWV 988 "Goldberg Variations": Aria',), ('Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude',), ('Toccata and Fugue in D Minor, BWV 565: I. Toccata',)]
    Answer:
    {'input': 'What are some example tracks by Bach?\nSQLQuery:SELECT "Name" FROM Track WHERE "Composer" LIKE \'%Bach%\' LIMIT 5;\nSQLResult: [(\'American Woman\',), (\'Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace\',), (\'Aria Mit 30 Veränderungen, BWV 988 "Goldberg Variations": Aria\',), (\'Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude\',), (\'Toccata and Fugue in D Minor, BWV 565: I. Toccata\',)]\nAnswer:', 'top_k': '5', 'dialect': 'sqlite', 'table_info': '\nCREATE TABLE "Playlist" (\n\t"PlaylistId" INTEGER NOT NULL, \n\t"Name" NVARCHAR(120), \n\tPRIMARY KEY ("PlaylistId")\n)\n\n/*\n2 rows from Playlist table:\nPlaylistId\tName\n1\tMusic\n2\tMovies\n*/\n\nCREATE TABLE Track (\n\t"TrackId" INTEGER NOT NULL, \n\t"Name" NVARCHAR(200) NOT NULL,\n\t"Composer" NVARCHAR(220),\n\tPRIMARY KEY ("TrackId")\n)\n/*\n3 rows from Track table:\nTrackId\tName\tComposer\n1\tFor Those About To Rock (We Salute You)\tAngus Young, Malcolm Young, Brian Johnson\n2\tBalls to the Wall\tNone\n3\tMy favorite song ever\tThe coolest composer of all time\n*/', 'stop': ['\nSQLResult:']}
    Examples of tracks by Bach include "American Woman", "Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace", "Aria Mit 30 Veränderungen, BWV 988 'Goldberg Variations': Aria", "Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude", and "Toccata and Fugue in D Minor, BWV 565: I. Toccata".
    > Finished chain.
```

```
    'Examples of tracks by Bach include "American Woman", "Concerto for 2 Violins in D Minor, BWV 1043: I. Vivace", "Aria Mit 30 Veränderungen, BWV 988 \'Goldberg Variations\': Aria", "Suite for Solo Cello No. 1 in G Major, BWV 1007: I. Prélude", and "Toccata and Fugue in D Minor, BWV 565: I. Toccata".'
```

SQLDatabaseSequentialChain

Chain for querying SQL database that is a sequential chain.

The chain is as follows:

```
1. Based on the query, determine which tables to use.
2. Based on those tables, call the normal SQL database chain.
```

This is useful in cases where the number of tables in the database is large.

```
from langchain.chains import SQLDatabaseSequentialChain
db = SQLDatabase.from_uri("sqlite:///../../../../notebooks/Chinook.db")
```

```
chain = SQLDatabaseSequentialChain.from_llm(llm, db, verbose=True)
```

```
chain.run("How many employees are also customers?")
```

```


    > Entering new SQLDatabaseSequentialChain chain...
    Table names to use:
    ['Employee', 'Customer']

    > Entering new SQLDatabaseChain chain...
```

```
How many employees are also customers?
SQLQuery:SELECT COUNT(*) FROM Employee e INNER JOIN Customer c ON e.EmployeeId = c.SupportRepId;
SQLResult: [(59,)]
Answer:59 employees are also customers.
> Finished chain.

> Finished chain.



'59 employees are also customers.'
```

Using Local Language Models

Sometimes you may not have the luxury of using OpenAI or other service-hosted large language model. You can, ofcourse, try to use the `SQLDatabaseChain` with a local model, but will quickly realize that most models you can run locally even with a large GPU struggle to generate the right output.

```
import logging
import torch
from transformers import AutoTokenizer, GPT2TokenizerFast, pipeline, AutoModelForSeq2SeqLM,
AutoModelForCausalLM
from langchain import HuggingFacePipeline

# Note: This model requires a large GPU, e.g. an 80GB A100. See documentation for other ways to run private
non-OpenAI models.
model_id = "google/flan-ul2"
```

```
model = AutoModelForSeq2SeqLM.from_pretrained(model_id, temperature=0)

device_id = -1  # default to no-GPU, but use GPU and half precision mode if available
if torch.cuda.is_available():
    device_id = 0
    try:
        model = model.half()
    except RuntimeError as exc:
        logging.warn(f"Could not run model in half precision mode: {str(exc)}")

tokenizer = AutoTokenizer.from_pretrained(model_id)
pipe = pipeline(task="text2text-generation", model=model, tokenizer=tokenizer, max_length=1024,
device=device_id)

local_llm = HuggingFacePipeline(pipeline=pipe)
```

```
    /workspace/langchain/.venv/lib/python3.9/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not found.
Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/en/stable/user_install.html
      from .autonotebook import tqdm as notebook_tqdm
    Loading checkpoint shards: 100%|██████████| 8/8 [00:32<00:00,  4.11s/it]
```

```
from langchain import SQLDatabase, SQLDatabaseChain

db = SQLDatabase.from_uri("sqlite:///../../../../notebooks/Chinook.db", include_tables=['Customer'])
local_chain = SQLDatabaseChain.from_llm(local_llm, db, verbose=True, return_intermediate_steps=True,
use_query_checker=True)
```

This model should work for very simple SQL queries, as long as you use the query checker as specified above, e.g.:

```
local_chain("How many customers are there?")
```

```
    > Entering new SQLDatabaseChain chain...
    How many customers are there?
    SQLQuery:

    /workspace/langchain/.venv/lib/python3.9/site-packages/transformers/pipelines/base.py:1070: UserWarning: You
sequentially on GPU. In order to maximize efficiency please use a dataset
      warnings.warn(
    /workspace/langchain/.venv/lib/python3.9/site-packages/transformers/pipelines/base.py:1070: UserWarning: You
sequentially on GPU. In order to maximize efficiency please use a dataset
      warnings.warn(


    SELECT count(*) FROM Customer
    SQLResult: [(59,)]
    Answer:

    /workspace/langchain/.venv/lib/python3.9/site-packages/transformers/pipelines/base.py:1070: UserWarning: You
sequentially on GPU. In order to maximize efficiency please use a dataset
      warnings.warn(


    [59]
    > Finished chain.
```

```
    {'query': 'How many customers are there?',
     'result': '[59]',
     'intermediate_steps': [{'input': 'How many customers are there?\nSQLQuery:SELECT count(*) FROM Customer\nSQ
       'top_k': '5',
       'dialect': 'sqlite',
       'table_info': '\nCREATE TABLE "Customer" (\n\t"CustomerId" INTEGER NOT NULL, \n\t"FirstName" NVARCHAR(40)
NOT NULL, \n\t"Company" NVARCHAR(80), \n\t"Address" NVARCHAR(70), \n\t"City" NVARCHAR(40), \n\t"State" NVARCHAR(
\n\t"PostalCode" NVARCHAR(10), \n\t"Phone" NVARCHAR(24), \n\t"Fax" NVARCHAR(24), \n\t"Email" NVARCHAR(60) NOT NU
\n\tPRIMARY KEY ("CustomerId"), \n\tFOREIGN KEY("SupportRepId") REFERENCES "Employee" ("EmployeeId")\n)\n\n/*\n3
table:\nCustomerId\tFirstName\tLastName\tCompany\tAddress\tCity\tState\tCountry\tPostalCode\tPhone\tFax\tEmail\t
- Empresa Brasileira de Aeronáutica S.A.\tAv. Brigadeiro Faria Lima, 2170\tSão José dos Campos\tSP\tBrazil\t122
5566\tluisg@embraer.com.br\t3\n2\tLeonie\tKöhler\tNone\tTheodor-Heuss-Straße 34\tStuttgart\tNone\tGermany\t70174
2842222\tNone\tleonekohler@surfeu.de\t5\n3\tFrançois\tTremblay\tNone\t1498 rue Bélanger\tMontréal\tQC\tCanada\tH
4711\tNone\tftremblay@gmail.com\t3\n*/',
       'stop': ['\nSQLResult:']},
      'SELECT count(*) FROM Customer',
      {'query': 'SELECT count(*) FROM Customer', 'dialect': 'sqlite'},
      'SELECT count(*) FROM Customer',
      '[(59,)]']}
```

Even this relatively large model will most likely fail to generate more complicated SQL by itself. However, you can log its inputs and outputs so that you can hand-correct them and use the corrected examples for few shot prompt examples later. In practice, you could log any executions of your chain that raise exceptions (as shown in the example below) or get direct user feedback in cases where the results are incorrect (but did not raise an exception).

```
poetry run pip install pyyaml chromadb
import yaml
```

```
    huggingface/tokenizers: The current process just got forked, after parallelism has already been used.
Disabling parallelism to avoid deadlocks...
    To disable this warning, you can either:
        - Avoid using `tokenizers` before the fork if possible
        - Explicitly set the environment variable TOKENIZERS_PARALLELISM=(true | false)


    11842.36s - pydevd: Sending message related to process being replaced timed-out after 5 seconds


    Requirement already satisfied: pyyaml in /workspace/langchain/.venv/lib/python3.9/site-packages (6.0)
    Requirement already satisfied: chromadb in /workspace/langchain/.venv/lib/python3.9/site-packages
(0.3.21)
    Requirement already satisfied: pandas>=1.3 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from chromadb) (2.0.1)
    Requirement already satisfied: requests>=2.28 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from chromadb) (2.28.2)
    Requirement already satisfied: pydantic>=1.9 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from chromadb) (1.10.7)
    Requirement already satisfied: hnswlib>=0.7 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from chromadb) (0.7.0)
    Requirement already satisfied: clickhouse-connect>=0.5.7 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from chromadb) (0.5.20)
    Requirement already satisfied: sentence-transformers>=2.2.2 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from chromadb) (2.2.2)
    Requirement already satisfied: duckdb>=0.7.1 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from chromadb) (0.7.1)
    Requirement already satisfied: fastapi>=0.85.1 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from chromadb) (0.95.1)
    Requirement already satisfied: uvicorn[standard]>=0.18.3 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from chromadb) (0.21.1)
    Requirement already satisfied: numpy>=1.21.6 in /workspace/langchain/.venv/lib/python3.9/site-packages
```

```
(from chromadb) (1.24.3)
    Requirement already satisfied: posthog>=2.4.0 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from chromadb) (3.0.1)
    Requirement already satisfied: certifi in /workspace/langchain/.venv/lib/python3.9/site-packages (from
clickhouse-connect>=0.5.7->chromadb) (2022.12.7)
    Requirement already satisfied: urllib3>=1.26 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from clickhouse-connect>=0.5.7->chromadb) (1.26.15)
    Requirement already satisfied: pytz in /workspace/langchain/.venv/lib/python3.9/site-packages (from
clickhouse-connect>=0.5.7->chromadb) (2023.3)
    Requirement already satisfied: zstandard in /workspace/langchain/.venv/lib/python3.9/site-packages (from
clickhouse-connect>=0.5.7->chromadb) (0.21.0)
    Requirement already satisfied: lz4 in /workspace/langchain/.venv/lib/python3.9/site-packages (from
clickhouse-connect>=0.5.7->chromadb) (4.3.2)
    Requirement already satisfied: starlette<0.27.0,>=0.26.1 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from fastapi>=0.85.1->chromadb) (0.26.1)
    Requirement already satisfied: python-dateutil>=2.8.2 in /workspace/langchain/.venv/lib/python3.9/site-
packages (from pandas>=1.3->chromadb) (2.8.2)
    Requirement already satisfied: tzdata>=2022.1 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from pandas>=1.3->chromadb) (2023.3)
    Requirement already satisfied: six>=1.5 in /workspace/langchain/.venv/lib/python3.9/site-packages (from
posthog>=2.4.0->chromadb) (1.16.0)
    Requirement already satisfied: monotonic>=1.5 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from posthog>=2.4.0->chromadb) (1.6)
    Requirement already satisfied: backoff>=1.10.0 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from posthog>=2.4.0->chromadb) (2.2.1)
    Requirement already satisfied: typing-extensions>=4.2.0 in /workspace/langchain/.venv/lib/python3.9/site-
packages (from pydantic>=1.9->chromadb) (4.5.0)
    Requirement already satisfied: charset-normalizer<4,>=2 in /workspace/langchain/.venv/lib/python3.9/site-
packages (from requests>=2.28->chromadb) (3.1.0)
    Requirement already satisfied: idna<4,>=2.5 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from requests>=2.28->chromadb) (3.4)
    Requirement already satisfied: transformers<5.0.0,>=4.6.0 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from sentence-transformers>=2.2.2->chromadb) (4.28.1)
```

```
    Requirement already satisfied: tqdm in /workspace/langchain/.venv/lib/python3.9/site-packages (from
sentence-transformers>=2.2.2->chromadb) (4.65.0)
    Requirement already satisfied: torch>=1.6.0 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from sentence-transformers>=2.2.2->chromadb) (1.13.1)
    Requirement already satisfied: torchvision in /workspace/langchain/.venv/lib/python3.9/site-packages
(from sentence-transformers>=2.2.2->chromadb) (0.14.1)
    Requirement already satisfied: scikit-learn in /workspace/langchain/.venv/lib/python3.9/site-packages
(from sentence-transformers>=2.2.2->chromadb) (1.2.2)
    Requirement already satisfied: scipy in /workspace/langchain/.venv/lib/python3.9/site-packages (from
sentence-transformers>=2.2.2->chromadb) (1.9.3)
    Requirement already satisfied: nltk in /workspace/langchain/.venv/lib/python3.9/site-packages (from
sentence-transformers>=2.2.2->chromadb) (3.8.1)
    Requirement already satisfied: sentencepiece in /workspace/langchain/.venv/lib/python3.9/site-packages
(from sentence-transformers>=2.2.2->chromadb) (0.1.98)
    Requirement already satisfied: huggingface-hub>=0.4.0 in /workspace/langchain/.venv/lib/python3.9/site-
packages (from sentence-transformers>=2.2.2->chromadb) (0.13.4)
    Requirement already satisfied: click>=7.0 in /workspace/langchain/.venv/lib/python3.9/site-packages (from
uvicorn[standard]>=0.18.3->chromadb) (8.1.3)
    Requirement already satisfied: h11>=0.8 in /workspace/langchain/.venv/lib/python3.9/site-packages (from
uvicorn[standard]>=0.18.3->chromadb) (0.14.0)
    Requirement already satisfied: httptools>=0.5.0 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from uvicorn[standard]>=0.18.3->chromadb) (0.5.0)
    Requirement already satisfied: python-dotenv>=0.13 in /workspace/langchain/.venv/lib/python3.9/site-
packages (from uvicorn[standard]>=0.18.3->chromadb) (1.0.0)
    Requirement already satisfied: uvloop!=0.15.0,!=0.15.1,>=0.14.0 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from uvicorn[standard]>=0.18.3->chromadb) (0.17.0)
    Requirement already satisfied: watchfiles>=0.13 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from uvicorn[standard]>=0.18.3->chromadb) (0.19.0)
    Requirement already satisfied: websockets>=10.4 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from uvicorn[standard]>=0.18.3->chromadb) (11.0.2)
    Requirement already satisfied: filelock in /workspace/langchain/.venv/lib/python3.9/site-packages (from
huggingface-hub>=0.4.0->sentence-transformers>=2.2.2->chromadb) (3.12.0)
    Requirement already satisfied: packaging>=20.9 in /workspace/langchain/.venv/lib/python3.9/site-packages
```

```
(from huggingface-hub>=0.4.0->sentence-transformers>=2.2.2->chromadb) (23.1)
    Requirement already satisfied: anyio<5,>=3.4.0 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from starlette<0.27.0,>=0.26.1->fastapi>=0.85.1->chromadb) (3.6.2)
    Requirement already satisfied: nvidia-cuda-runtime-cu11==11.7.99 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from torch>=1.6.0->sentence-transformers>=2.2.2-
>chromadb) (11.7.99)
    Requirement already satisfied: nvidia-cudnn-cu11==8.5.0.96 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from torch>=1.6.0->sentence-transformers>=2.2.2-
>chromadb) (8.5.0.96)
    Requirement already satisfied: nvidia-cublas-cu11==11.10.3.66 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from torch>=1.6.0->sentence-transformers>=2.2.2-
>chromadb) (11.10.3.66)
    Requirement already satisfied: nvidia-cuda-nvrtc-cu11==11.7.99 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from torch>=1.6.0->sentence-transformers>=2.2.2-
>chromadb) (11.7.99)
    Requirement already satisfied: setuptools in /workspace/langchain/.venv/lib/python3.9/site-packages (from
nvidia-cublas-cu11==11.10.3.66->torch>=1.6.0->sentence-transformers>=2.2.2->chromadb) (67.7.1)
    Requirement already satisfied: wheel in /workspace/langchain/.venv/lib/python3.9/site-packages (from
nvidia-cublas-cu11==11.10.3.66->torch>=1.6.0->sentence-transformers>=2.2.2->chromadb) (0.40.0)
    Requirement already satisfied: regex!=2019.12.17 in /workspace/langchain/.venv/lib/python3.9/site-
packages (from transformers<5.0.0,>=4.6.0->sentence-transformers>=2.2.2->chromadb) (2023.3.23)
    Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in
/workspace/langchain/.venv/lib/python3.9/site-packages (from transformers<5.0.0,>=4.6.0->sentence-
transformers>=2.2.2->chromadb) (0.13.3)
    Requirement already satisfied: joblib in /workspace/langchain/.venv/lib/python3.9/site-packages (from
nltk->sentence-transformers>=2.2.2->chromadb) (1.2.0)
    Requirement already satisfied: threadpoolctl>=2.0.0 in /workspace/langchain/.venv/lib/python3.9/site-
packages (from scikit-learn->sentence-transformers>=2.2.2->chromadb) (3.1.0)
    Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /workspace/langchain/.venv/lib/python3.9/site-
packages (from torchvision->sentence-transformers>=2.2.2->chromadb) (9.5.0)
    Requirement already satisfied: sniffio>=1.1 in /workspace/langchain/.venv/lib/python3.9/site-packages
(from anyio<5,>=3.4.0->starlette<0.27.0,>=0.26.1->fastapi>=0.85.1->chromadb) (1.3.0)
```

```
from typing import Dict

QUERY = "List all the customer first names that start with 'a'"

def _parse_example(result: Dict) -> Dict:
    sql_cmd_key = "sql_cmd"
    sql_result_key = "sql_result"
    table_info_key = "table_info"
    input_key = "input"
    final_answer_key = "answer"

    _example = {
        "input": result.get("query"),
    }

    steps = result.get("intermediate_steps")
    answer_key = sql_cmd_key # the first one
    for step in steps:
        # The steps are in pairs, a dict (input) followed by a string (output).
        # Unfortunately there is no schema but you can look at the input key of the
        # dict to see what the output is supposed to be
        if isinstance(step, dict):
            # Grab the table info from input dicts in the intermediate steps once
            if table_info_key not in _example:
                _example[table_info_key] = step.get(table_info_key)

            if input_key in step:
                if step[input_key].endswith("SQLQuery:"):
                    answer_key = sql_cmd_key # this is the SQL generation input
                if step[input_key].endswith("Answer:"):
                    answer_key = final_answer_key # this is the final answer input
            elif sql_cmd_key in step:
```

```
                _example[sql_cmd_key] = step[sql_cmd_key]
                answer_key = sql_result_key # this is SQL execution input
        elif isinstance(step, str):
            # The preceding element should have set the answer_key
            _example[answer_key] = step
    return _example

example: any
try:
    result = local_chain(QUERY)
    print("*** Query succeeded")
    example = _parse_example(result)
except Exception as exc:
    print("*** Query failed")
    result = {
        "query": QUERY,
        "intermediate_steps": exc.intermediate_steps
    }
    example = _parse_example(result)


# print for now, in reality you may want to write this out to a YAML file or database for manual fix-ups
offline
yaml_example = yaml.dump(example, allow_unicode=True)
print("\n" + yaml_example)
```

```
    > Entering new SQLDatabaseChain chain...
    List all the customer first names that start with 'a'
    SQLQuery:
```

```
    /workspace/langchain/.venv/lib/python3.9/site-packages/transformers/pipelines/base.py:1070: UserWarning:
You seem to be using the pipelines sequentially on GPU. In order to maximize efficiency please use a dataset
      warnings.warn(


    SELECT firstname FROM customer WHERE firstname LIKE '%a%'
    SQLResult: [('François',), ('František',), ('Helena',), ('Astrid',), ('Daan',), ('Kara',), ('Eduardo',),
('Alexandre',), ('Fernanda',), ('Mark',), ('Frank',), ('Jack',), ('Dan',), ('Kathy',), ('Heather',),
('Frank',), ('Richard',), ('Patrick',), ('Julia',), ('Edward',), ('Martha',), ('Aaron',), ('Madalena',),
('Hannah',), ('Niklas',), ('Camille',), ('Marc',), ('Wyatt',), ('Isabelle',), ('Ladislav',), ('Lucas',),
('Johannes',), ('Stanisław',), ('Joakim',), ('Emma',), ('Mark',), ('Manoj',), ('Puja',)]
    Answer:

    /workspace/langchain/.venv/lib/python3.9/site-packages/transformers/pipelines/base.py:1070: UserWarning:
You seem to be using the pipelines sequentially on GPU. In order to maximize efficiency please use a dataset
      warnings.warn(


    [('François', 'Frantiek', 'Helena', 'Astrid', 'Daan', 'Kara', 'Eduardo', 'Alexandre', 'Fernanda', 'Mark',
'Frank', 'Jack', 'Dan', 'Kathy', 'Heather', 'Frank', 'Richard', 'Patrick', 'Julia', 'Edward', 'Martha',
'Aaron', 'Madalena', 'Hannah', 'Niklas', 'Camille', 'Marc', 'Wyatt', 'Isabelle', 'Ladislav', 'Lucas',
'Johannes', 'Stanisaw', 'Joakim', 'Emma', 'Mark', 'Manoj', 'Puja']
    > Finished chain.
    *** Query succeeded

    answer: '[(''François'', ''Frantiek'', ''Helena'', ''Astrid'', ''Daan'', ''Kara'',
      ''Eduardo'', ''Alexandre'', ''Fernanda'', ''Mark'', ''Frank'', ''Jack'', ''Dan'',
      ''Kathy'', ''Heather'', ''Frank'', ''Richard'', ''Patrick'', ''Julia'', ''Edward'',
      ''Martha'', ''Aaron'', ''Madalena'', ''Hannah'', ''Niklas'', ''Camille'', ''Marc'',
      ''Wyatt'', ''Isabelle'', ''Ladislav'', ''Lucas'', ''Johannes'', ''Stanisaw'', ''Joakim'',
      ''Emma'', ''Mark'', ''Manoj'', ''Puja'']'
    input: List all the customer first names that start with 'a'
    sql_cmd: SELECT firstname FROM customer WHERE firstname LIKE '%a%'
```

```
    sql_result: '[(''François'',), (''František'',), (''Helena'',), (''Astrid'',), (''Daan'',),
      (''Kara'',), (''Eduardo'',), (''Alexandre'',), (''Fernanda'',), (''Mark'',), (''Frank'',),
      (''Jack'',), (''Dan'',), (''Kathy'',), (''Heather'',), (''Frank'',), (''Richard'',),
      (''Patrick'',), (''Julia'',), (''Edward'',), (''Martha'',), (''Aaron'',), (''Madalena'',),
      (''Hannah'',), (''Niklas'',), (''Camille'',), (''Marc'',), (''Wyatt'',), (''Isabelle'',),
      (''Ladislav'',), (''Lucas'',), (''Johannes'',), (''Stanisław'',), (''Joakim'',),
      (''Emma'',), (''Mark'',), (''Manoj'',), (''Puja'',)]'
    table_info: "\nCREATE TABLE \"Customer\" (\n\t\"CustomerId\" INTEGER NOT NULL, \n\t\
      \"FirstName\" NVARCHAR(40) NOT NULL, \n\t\"LastName\" NVARCHAR(20) NOT NULL, \n\t\
      \"Company\" NVARCHAR(80), \n\t\"Address\" NVARCHAR(70), \n\t\"City\" NVARCHAR(40),\
      \ \n\t\"State\" NVARCHAR(40), \n\t\"Country\" NVARCHAR(40), \n\t\"PostalCode\" NVARCHAR(10),\
      \ \n\t\"Phone\" NVARCHAR(24), \n\t\"Fax\" NVARCHAR(24), \n\t\"Email\" NVARCHAR(60)\
      \ NOT NULL, \n\t\"SupportRepId\" INTEGER, \n\tPRIMARY KEY (\"CustomerId\"), \n\t\
      FOREIGN KEY(\"SupportRepId\") REFERENCES \"Employee\" (\"EmployeeId\")\n)\n\n/*\n\
      3 rows from Customer table:\nCustomerId\tFirstName\tLastName\tCompany\tAddress\t\
      City\tState\tCountry\tPostalCode\tPhone\tFax\tEmail\tSupportRepId\n1\tLuís\tGonçalves\t\
      Embraer - Empresa Brasileira de Aeronáutica S.A.\tAv. Brigadeiro Faria Lima, 2170\t\
      São José dos Campos\tSP\tBrazil\t12227-000\t+55 (12) 3923-5555\t+55 (12) 3923-5566\t\
      luisg@embraer.com.br\t3\n2\tLeonie\tKöhler\tNone\tTheodor-Heuss-Straße 34\tStuttgart\t\
      None\tGermany\t70174\t+49 0711 2842222\tNone\tleonekohler@surfeu.de\t5\n3\tFrançois\t\
      Tremblay\tNone\t1498 rue Bélanger\tMontréal\tQC\tCanada\tH2G 1A7\t+1 (514) 721-4711\t\
      None\tftremblay@gmail.com\t3\n*/"
```

Run the snippet above a few times, or log exceptions in your deployed environment, to collect lots of examples of inputs, table_info and sql_cmd generated by your language model. The sql_cmd values will be incorrect and you can manually fix them up to build a collection of examples, e.g. here we are using YAML to keep a neat record of our inputs and corrected SQL output that we can build up over time.

```
YAML_EXAMPLES = """
- input: How many customers are not from Brazil?
  table_info: |
```

```
    CREATE TABLE "Customer" (
      "CustomerId" INTEGER NOT NULL,
      "FirstName" NVARCHAR(40) NOT NULL,
      "LastName" NVARCHAR(20) NOT NULL,
      "Company" NVARCHAR(80),
      "Address" NVARCHAR(70),
      "City" NVARCHAR(40),
      "State" NVARCHAR(40),
      "Country" NVARCHAR(40),
      "PostalCode" NVARCHAR(10),
      "Phone" NVARCHAR(24),
      "Fax" NVARCHAR(24),
      "Email" NVARCHAR(60) NOT NULL,
      "SupportRepId" INTEGER,
      PRIMARY KEY ("CustomerId"),
      FOREIGN KEY("SupportRepId") REFERENCES "Employee" ("EmployeeId")
    )
  sql_cmd: SELECT COUNT(*) FROM "Customer" WHERE NOT "Country" = "Brazil";
  sql_result: "[(54,)]"
  answer: 54 customers are not from Brazil.
- input: list all the genres that start with 'r'
  table_info: |
    CREATE TABLE "Genre" (
      "GenreId" INTEGER NOT NULL,
      "Name" NVARCHAR(120),
      PRIMARY KEY ("GenreId")
    )

    /*
    3 rows from Genre table:
    GenreId Name
    1   Rock
    2   Jazz
```

```
    3   Metal
    */
  sql_cmd: SELECT "Name" FROM "Genre" WHERE "Name" LIKE 'r%';
  sql_result: "[('Rock',), ('Rock and Roll',), ('Reggae',), ('R&B/Soul',)]"
  answer: The genres that start with 'r' are Rock, Rock and Roll, Reggae and R&B/Soul.
"""
```

Now that you have some examples (with manually corrected output SQL), you can do few shot prompt seeding the usual way:

```
from langchain import FewShotPromptTemplate, PromptTemplate
from langchain.chains.sql_database.prompt import _sqlite_prompt, PROMPT_SUFFIX
from langchain.embeddings.huggingface import HuggingFaceEmbeddings
from langchain.prompts.example_selector.semantic_similarity import SemanticSimilarityExampleSelector
from langchain.vectorstores import Chroma

example_prompt = PromptTemplate(
    input_variables=["table_info", "input", "sql_cmd", "sql_result", "answer"],
    template="{table_info}\n\nQuestion: {input}\nSQLQuery: {sql_cmd}\nSQLResult: {sql_result}\nAnswer:
{answer}",
)

examples_dict = yaml.safe_load(YAML_EXAMPLES)

local_embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")

example_selector = SemanticSimilarityExampleSelector.from_examples(
                        # This is the list of examples available to select from.
                        examples_dict,
                        # This is the embedding class used to produce embeddings which are used to measure
semantic similarity.
                        local_embeddings,
                        # This is the VectorStore class that is used to store the embeddings and do a
```

```
similarity search over.
                        Chroma,  # type: ignore
                        # This is the number of examples to produce and include per prompt
                        k=min(3, len(examples_dict)),
                    )

few_shot_prompt = FewShotPromptTemplate(
    example_selector=example_selector,
    example_prompt=example_prompt,
    prefix=_sqlite_prompt + "Here are some examples:",
    suffix=PROMPT_SUFFIX,
    input_variables=["table_info", "input", "top_k"],
)
```

```
    Using embedded DuckDB without persistence: data will be transient
```

The model should do better now with this few shot prompt, especially for inputs similar to the examples you have seeded it with.

```
local_chain = SQLDatabaseChain.from_llm(local_llm, db, prompt=few_shot_prompt, use_query_checker=True,
verbose=True, return_intermediate_steps=True)
```

```
result = local_chain("How many customers are from Brazil?")
```

```


    > Entering new SQLDatabaseChain chain...
    How many customers are from Brazil?
    SQLQuery:SELECT count(*) FROM Customer WHERE Country = "Brazil";
```

```
    SQLResult: [(5,)]
    Answer:[5]
    > Finished chain.
```

```
result = local_chain("How many customers are not from Brazil?")
```

```
    > Entering new SQLDatabaseChain chain...
    How many customers are not from Brazil?
    SQLQuery:SELECT count(*) FROM customer WHERE country NOT IN (SELECT country FROM customer WHERE country =
'Brazil')
    SQLResult: [(54,)]
    Answer:54 customers are not from Brazil.
    > Finished chain.
```

```
result = local_chain("How many customers are there in total?")
```

```
    > Entering new SQLDatabaseChain chain...
    How many customers are there in total?
    SQLQuery:SELECT count(*) FROM Customer;
    SQLResult: [(59,)]
    Answer:There are 59 customers in total.
    > Finished chain.
```