SQL Database Agent

This notebook showcases an agent designed to interact with a sql databases. The agent builds off of SQLDatabaseChain and is designed to answer more general questions about a database, as well as recover from errors.

Note that, as this agent is in active development, all answers might not be correct. Additionally, it is not guaranteed that the agent won't perform DML statements on your database given certain questions. Be careful running it on sensitive data!

This uses 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.

Initialization

```
from langchain.agents import create_sql_agent
from langchain.agents.agent_toolkits import SQLDatabaseToolkit
from langchain.sql_database import SQLDatabase
from langchain.llms.openai import OpenAI
from langchain.agents import AgentExecutor
from langchain.agents.agent_types import AgentType
from langchain.chat_models import ChatOpenAI
```

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

Using ZERO_SHOT_REACT_DESCRIPTION

This shows how to initialize the agent using the ZERO_SHOT_REACT_DESCRIPTION agent type. Note that this is an alternative to the above.

```
agent_executor = create_sql_agent(
    llm=OpenAI(temperature=0),
    toolkit=toolkit,
    verbose=True,
    agent_type=AgentType.ZERO_SHOT_REACT_DESCRIPTION,
)
```

Using OpenAI Functions

This shows how to initialize the agent using the OPENAI_FUNCTIONS agent type. Note that this is an alternative to the above.

```
# agent_executor = create_sql_agent(
#     llm=ChatOpenAI(temperature=0, model="gpt-3.5-turbo-0613"),
#     toolkit=toolkit,
#     verbose=True,
#     agent_type=AgentType.OPENAI_FUNCTIONS
# )
```

Example: describing a table

```
agent_executor.run("Describe the playlisttrack table")
```

```
    > Entering new  chain...

    Invoking: `list_tables_sql_db` with `{}`


    Album, Artist, Track, PlaylistTrack, InvoiceLine, sales_table, Playlist, Genre, Employee, Customer,
Invoice, MediaType
    Invoking: `schema_sql_db` with `PlaylistTrack`



    CREATE TABLE "PlaylistTrack" (
        "PlaylistId" INTEGER NOT NULL,
        "TrackId" INTEGER NOT NULL,
        PRIMARY KEY ("PlaylistId", "TrackId"),
        FOREIGN KEY("TrackId") REFERENCES "Track" ("TrackId"),
        FOREIGN KEY("PlaylistId") REFERENCES "Playlist" ("PlaylistId")
    )

    /*
    3 rows from PlaylistTrack table:
    PlaylistId  TrackId
    1   3402
    1   3389
    1   3390
    */The `PlaylistTrack` table has two columns: `PlaylistId` and `TrackId`. It is a junction table that
represents the relationship between playlists and tracks.
```

```
Here is the schema of the `PlaylistTrack` table:

```
CREATE TABLE "PlaylistTrack" (
    "PlaylistId" INTEGER NOT NULL,
    "TrackId" INTEGER NOT NULL,
    PRIMARY KEY ("PlaylistId", "TrackId"),
    FOREIGN KEY("TrackId") REFERENCES "Track" ("TrackId"),
    FOREIGN KEY("PlaylistId") REFERENCES "Playlist" ("PlaylistId")
)
```

Here are three sample rows from the `PlaylistTrack` table:

```
PlaylistId   TrackId
1            3402
1            3389
1            3390
```

Please let me know if there is anything else I can help you with.

> Finished chain.
```

```
'The `PlaylistTrack` table has two columns: `PlaylistId` and `TrackId`. It is a junction table that represents the relationship between playlists and tracks. \n\nHere is the schema of the `PlaylistTrack` table:\n\n```\nCREATE TABLE "PlaylistTrack" (\n\t"PlaylistId" INTEGER NOT NULL, \n\t"TrackId" INTEGER NOT
```

```
NULL, \n\tPRIMARY KEY ("PlaylistId", "TrackId"), \n\tFOREIGN KEY("TrackId") REFERENCES "Track" ("TrackId"),
\n\tFOREIGN KEY("PlaylistId") REFERENCES "Playlist" ("PlaylistId")\n)\n```\n\nHere are three sample rows from
the `PlaylistTrack` table:\n\n```\nPlaylistId   TrackId\n1            3402\n1            3389\n1
3390\n```\n\nPlease let me know if there is anything else I can help you with.'
```

Example: describing a table, recovering from an error

In this example, the agent tries to search for a table that doesn't exist, but finds the next best result

```
agent_executor.run("Describe the playlistsong table")
```

```
    > Entering new AgentExecutor chain...
    Action: list_tables_sql_db
    Action Input: ""
    Observation: Genre, PlaylistTrack, MediaType, Invoice, InvoiceLine, Track, Playlist, Customer, Album,
Employee, Artist
    Thought: I should look at the schema of the PlaylistSong table
    Action: schema_sql_db
    Action Input: "PlaylistSong"
    Observation: Error: table_names {'PlaylistSong'} not found in database
    Thought: I should check the spelling of the table
    Action: list_tables_sql_db
    Action Input: ""
    Observation: Genre, PlaylistTrack, MediaType, Invoice, InvoiceLine, Track, Playlist, Customer, Album,
Employee, Artist
    Thought: The table is called PlaylistTrack
```

```
    Action: schema_sql_db
    Action Input: "PlaylistTrack"
    Observation:
    CREATE TABLE "PlaylistTrack" (
        "PlaylistId" INTEGER NOT NULL,
        "TrackId" INTEGER NOT NULL,
        PRIMARY KEY ("PlaylistId", "TrackId"),
        FOREIGN KEY("TrackId") REFERENCES "Track" ("TrackId"),
        FOREIGN KEY("PlaylistId") REFERENCES "Playlist" ("PlaylistId")
    )

    SELECT * FROM 'PlaylistTrack' LIMIT 3;
    PlaylistId TrackId
    1 3402
    1 3389
    1 3390
    Thought: I now know the final answer
    Final Answer: The PlaylistTrack table contains two columns, PlaylistId and TrackId, which are both
integers and are used to link Playlist and Track tables.

    > Finished chain.




    'The PlaylistTrack table contains two columns, PlaylistId and TrackId, which are both integers and are
used to link Playlist and Track tables.'
```

Example: running queries

```
agent_executor.run(
    "List the total sales per country. Which country's customers spent the most?"
)
```

```
    > Entering new AgentExecutor chain...
    Action: list_tables_sql_db
    Action Input: ""
    Observation: Invoice, MediaType, Artist, InvoiceLine, Genre, Playlist, Employee, Album, PlaylistTrack,
Track, Customer
    Thought: I should look at the schema of the relevant tables to see what columns I can use.
    Action: schema_sql_db
    Action Input: "Invoice, Customer"
    Observation:
    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")
```

```
    )

    SELECT * FROM 'Customer' LIMIT 3;
    CustomerId FirstName LastName Company Address City State Country PostalCode Phone Fax Email SupportRepId
    1 Luís Gonçalves Embraer - Empresa Brasileira de Aeronáutica S.A. Av. Brigadeiro Faria Lima, 2170 São José dos Campos SP Brazil 12227-000 +55 (12) 3923-5555 +55 (12) 3923-5566 luisg@embraer.com.br 3
    2 Leonie Köhler None Theodor-Heuss-Straße 34 Stuttgart None Germany 70174 +49 0711 2842222 None leonekohler@surfeu.de 5
    3 François Tremblay None 1498 rue Bélanger Montréal QC Canada H2G 1A7 +1 (514) 721-4711 None ftremblay@gmail.com 3


    CREATE TABLE "Invoice" (
        "InvoiceId" INTEGER NOT NULL,
        "CustomerId" INTEGER NOT NULL,
        "InvoiceDate" DATETIME NOT NULL,
        "BillingAddress" NVARCHAR(70),
        "BillingCity" NVARCHAR(40),
        "BillingState" NVARCHAR(40),
        "BillingCountry" NVARCHAR(40),
        "BillingPostalCode" NVARCHAR(10),
        "Total" NUMERIC(10, 2) NOT NULL,
        PRIMARY KEY ("InvoiceId"),
        FOREIGN KEY("CustomerId") REFERENCES "Customer" ("CustomerId")
    )

    SELECT * FROM 'Invoice' LIMIT 3;
    InvoiceId CustomerId InvoiceDate BillingAddress BillingCity BillingState BillingCountry BillingPostalCode Total
    1 2 2009-01-01 00:00:00 Theodor-Heuss-Straße 34 Stuttgart None Germany 70174 1.98
    2 4 2009-01-02 00:00:00 Ullevålsveien 14 Oslo None Norway 0171 3.96
    3 8 2009-01-03 00:00:00 Grétrystraat 63 Brussels None Belgium 1000 5.94
    Thought: I should query the Invoice and Customer tables to get the total sales per country.
```

```
    Action: query_sql_db
    Action Input: SELECT c.Country, SUM(i.Total) AS TotalSales FROM Invoice i INNER JOIN Customer c ON
i.CustomerId = c.CustomerId GROUP BY c.Country ORDER BY TotalSales DESC LIMIT 10
    Observation: [('USA', 523.0600000000003), ('Canada', 303.9599999999999), ('France', 195.09999999999994),
('Brazil', 190.09999999999997), ('Germany', 156.48), ('United Kingdom', 112.85999999999999), ('Czech
Republic', 90.24000000000001), ('Portugal', 77.23999999999998), ('India', 75.25999999999999), ('Chile',
46.62)]
    Thought: I now know the final answer
    Final Answer: The customers from the USA spent the most, with a total of $523.06.

    > Finished chain.
```

```
    'The customers from the USA spent the most, with a total of $523.06.'
```

```
agent_executor.run(
    "Show the total number of tracks in each playlist. The Playlist name should be included in the result."
)
```

```
    > Entering new AgentExecutor chain...
    Action: list_tables_sql_db
    Action Input: ""
    Observation: Invoice, MediaType, Artist, InvoiceLine, Genre, Playlist, Employee, Album, PlaylistTrack,
Track, Customer
    Thought: I should look at the schema of the Playlist and PlaylistTrack tables to see what columns I can
use.
```

```
    Action: schema_sql_db
    Action Input: "Playlist, PlaylistTrack"
    Observation:
    CREATE TABLE "Playlist" (
        "PlaylistId" INTEGER NOT NULL,
        "Name" NVARCHAR(120),
        PRIMARY KEY ("PlaylistId")
    )

    SELECT * FROM 'Playlist' LIMIT 3;
    PlaylistId Name
    1 Music
    2 Movies
    3 TV Shows



    CREATE TABLE "PlaylistTrack" (
        "PlaylistId" INTEGER NOT NULL,
        "TrackId" INTEGER NOT NULL,
        PRIMARY KEY ("PlaylistId", "TrackId"),
        FOREIGN KEY("TrackId") REFERENCES "Track" ("TrackId"),
        FOREIGN KEY("PlaylistId") REFERENCES "Playlist" ("PlaylistId")
    )

    SELECT * FROM 'PlaylistTrack' LIMIT 3;
    PlaylistId TrackId
    1 3402
    1 3389
    1 3390
    Thought: I can use a SELECT statement to get the total number of tracks in each playlist.
    Action: query_checker_sql_db
    Action Input: SELECT Playlist.Name, COUNT(PlaylistTrack.TrackId) AS TotalTracks FROM Playlist INNER JOIN
PlaylistTrack ON Playlist.PlaylistId = PlaylistTrack.PlaylistId GROUP BY Playlist.Name
```

```
    Observation:

    SELECT Playlist.Name, COUNT(PlaylistTrack.TrackId) AS TotalTracks FROM Playlist INNER JOIN PlaylistTrack
ON Playlist.PlaylistId = PlaylistTrack.PlaylistId GROUP BY Playlist.Name
    Thought: The query looks correct, I can now execute it.
    Action: query_sql_db
    Action Input: SELECT Playlist.Name, COUNT(PlaylistTrack.TrackId) AS TotalTracks FROM Playlist INNER JOIN
PlaylistTrack ON Playlist.PlaylistId = PlaylistTrack.PlaylistId GROUP BY Playlist.Name LIMIT 10
    Observation: [('90’s Music', 1477), ('Brazilian Music', 39), ('Classical', 75), ('Classical 101 - Deep
Cuts', 25), ('Classical 101 - Next Steps', 25), ('Classical 101 - The Basics', 25), ('Grunge', 15), ('Heavy
Metal Classic', 26), ('Music', 6580), ('Music Videos', 1)]
    Thought: I now know the final answer.
    Final Answer: The total number of tracks in each playlist are: '90’s Music' (1477), 'Brazilian Music'
(39), 'Classical' (75), 'Classical 101 - Deep Cuts' (25), 'Classical 101 - Next Steps' (25), 'Classical 101 -
The Basics' (25), 'Grunge' (15), 'Heavy Metal Classic' (26), 'Music' (6580), 'Music Videos' (1).

    > Finished chain.




    "The total number of tracks in each playlist are: '90’s Music' (1477), 'Brazilian Music' (39),
'Classical' (75), 'Classical 101 - Deep Cuts' (25), 'Classical 101 - Next Steps' (25), 'Classical 101 - The
Basics' (25), 'Grunge' (15), 'Heavy Metal Classic' (26), 'Music' (6580), 'Music Videos' (1)."
```

Recovering from an error

In this example, the agent is able to recover from an error after initially trying to access an attribute (`Track.ArtistId`) which doesn't exist.

```
agent_executor.run("Who are the top 3 best selling artists?")
```

```
    > Entering new AgentExecutor chain...
    Action: list_tables_sql_db
    Action Input: ""
    Observation: MediaType, Track, Invoice, Album, Playlist, Customer, Employee, InvoiceLine, PlaylistTrack,
Genre, Artist
    Thought: I should look at the schema of the Artist, InvoiceLine, and Track tables to see what columns I
can use.
    Action: schema_sql_db
    Action Input: "Artist, InvoiceLine, Track"
    Observation:
    CREATE TABLE "Artist" (
        "ArtistId" INTEGER NOT NULL,
        "Name" NVARCHAR(120),
        PRIMARY KEY ("ArtistId")
    )

    SELECT * FROM 'Artist' LIMIT 3;
    ArtistId Name
    1 AC/DC
    2 Accept
    3 Aerosmith


    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")
    )

    SELECT * FROM 'Track' LIMIT 3;
    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
    3 Fast As a Shark 3 2 1 F. Baltes, S. Kaufman, U. Dirkscneider & W. Hoffman 230619 3990994 0.99


    CREATE TABLE "InvoiceLine" (
        "InvoiceLineId" INTEGER NOT NULL,
        "InvoiceId" INTEGER NOT NULL,
        "TrackId" INTEGER NOT NULL,
        "UnitPrice" NUMERIC(10, 2) NOT NULL,
        "Quantity" INTEGER NOT NULL,
        PRIMARY KEY ("InvoiceLineId"),
        FOREIGN KEY("TrackId") REFERENCES "Track" ("TrackId"),
        FOREIGN KEY("InvoiceId") REFERENCES "Invoice" ("InvoiceId")
    )

    SELECT * FROM 'InvoiceLine' LIMIT 3;
    InvoiceLineId InvoiceId TrackId UnitPrice Quantity
```

```
    1 1 2 0.99 1
    2 1 4 0.99 1
    3 2 6 0.99 1
    Thought: I should query the database to get the top 3 best selling artists.
    Action: query_sql_db
    Action Input: SELECT Artist.Name, SUM(InvoiceLine.Quantity) AS TotalQuantity FROM Artist INNER JOIN Track
ON Artist.ArtistId = Track.ArtistId INNER JOIN InvoiceLine ON Track.TrackId = InvoiceLine.TrackId GROUP BY
Artist.Name ORDER BY TotalQuantity DESC LIMIT 3
    Observation: Error: (sqlite3.OperationalError) no such column: Track.ArtistId
    [SQL: SELECT Artist.Name, SUM(InvoiceLine.Quantity) AS TotalQuantity FROM Artist INNER JOIN Track ON
Artist.ArtistId = Track.ArtistId INNER JOIN InvoiceLine ON Track.TrackId = InvoiceLine.TrackId GROUP BY
Artist.Name ORDER BY TotalQuantity DESC LIMIT 3]
    (Background on this error at: https://sqlalche.me/e/14/e3q8)
    Thought: I should double check my query before executing it.
    Action: query_checker_sql_db
    Action Input: SELECT Artist.Name, SUM(InvoiceLine.Quantity) AS TotalQuantity FROM Artist INNER JOIN Track
ON Artist.ArtistId = Track.ArtistId INNER JOIN InvoiceLine ON Track.TrackId = InvoiceLine.TrackId GROUP BY
Artist.Name ORDER BY TotalQuantity DESC LIMIT 3
    Observation:

    SELECT Artist.Name, SUM(InvoiceLine.Quantity) AS TotalQuantity
    FROM Artist
    INNER JOIN Track ON Artist.ArtistId = Track.ArtistId
    INNER JOIN InvoiceLine ON Track.TrackId = InvoiceLine.TrackId
    GROUP BY Artist.Name
    ORDER BY TotalQuantity DESC
    LIMIT 3;
    Thought: I now know the final answer.
    Action: query_sql_db
    Action Input: SELECT Artist.Name, SUM(InvoiceLine.Quantity) AS TotalQuantity FROM Artist INNER JOIN Album
ON Artist.ArtistId = Album.ArtistId INNER JOIN Track ON Album.AlbumId = Track.AlbumId INNER JOIN InvoiceLine
ON Track.TrackId = InvoiceLine.TrackId GROUP BY Artist.Name ORDER BY TotalQuantity DESC LIMIT 3
    Observation: [('Iron Maiden', 140), ('U2', 107), ('Metallica', 91)]
```

```
Thought: I now know the final answer.
Final Answer: The top 3 best selling artists are Iron Maiden, U2, and Metallica.

> Finished chain.
```

```
'The top 3 best selling artists are Iron Maiden, U2, and Metallica.'
```