```
        location: str

        class Config:
            arbitrary_types_allowed = True
            schema_extra = {
                "example": {
                    "title": "FastAPI Book Launch",
                    "image": "https:
                    //linktomyimage.com/image.png",
                    "description": "We will be discussing
                    the contents of the FastAPI book in
                    this event. Ensure to come with your
                    own copy to win gifts!",
                    "tags": ["python", "fastapi", "book",
                    "launch"],
                    "location": "Google Meet"
                }
            }
```

In this code block, we have modified the original model class to become a SQL table class.

2. Let's add another SQLModel class that'll be used as the body type during `UPDATE` operations:

```
    class EventUpdate(SQLModel):
        title: Optional[str]
        image: Optional[str]
        description: Optional[str]
        tags: Optional[List[str]]
        location: Optional[str]

        class Config:
            schema_extra = {
                "example": {
                    "title": "FastAPI Book Launch",
                    "image": "https:
```

```
                  //linktomyimage.com/image.png",
                  "description": "We will be discussing
                  the contents of the FastAPI book in
                  this event. Ensure to come with your
                  own copy to win gifts!",
                  "tags": ["python", "fastapi", "book",
                  "launch"],
                  "location": "Google Meet"
              }
          }
```

3. Next, let's define the configuration needed to create our database and table in `connection.py`:

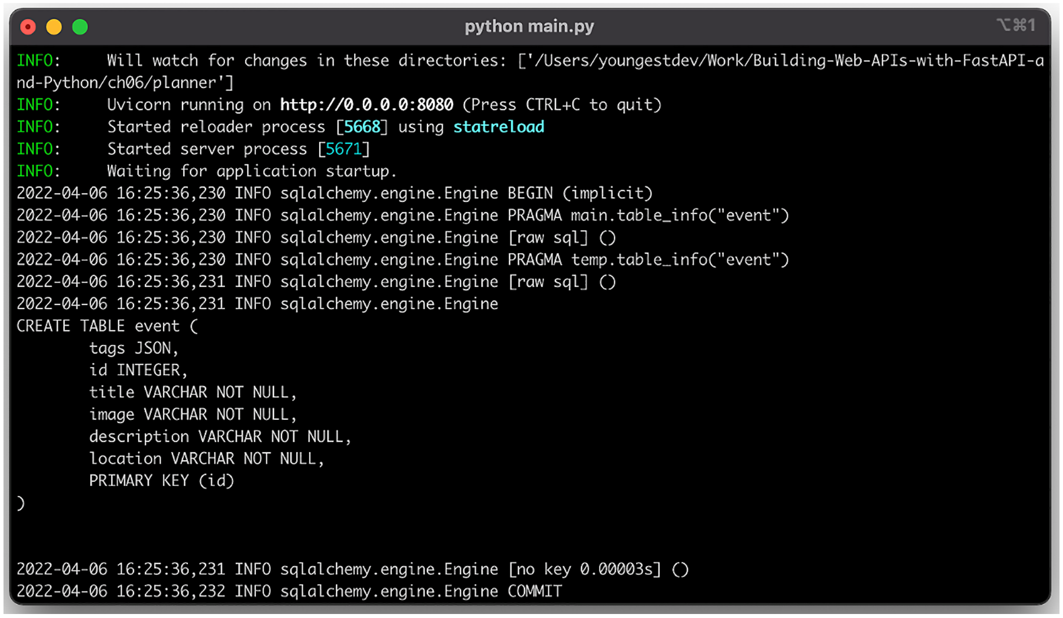
```
from sqlmodel import SQLModel, Session, create_engine
from models.events import Event

database_file = "planner.db"
database_connection_string = f"sqlite:///{database_file}"
connect_args = {"check_same_thread": False}
engine_url = create_engine(database_connection_string,
echo=True, connect_args=connect_args)

def conn():
    SQLModel.metadata.create_all(engine_url)

def get_session():
    with Session(engine_url) as session:
        yield session
```

In this code block, we start by defining the dependencies as well as importing the table model class. Next, we create the variable holding the location of the database file (which will be created if it doesn't exist), the connection string, and an instance of the SQL database created. In the `conn()` function, we instruct SQLModel to create the database as well as the table present in the file, `Events`, and to persist the session in our application, `get_session()` is defined.

4. Next, let's instruct our application to create a database when it is started. Update `main.py` with the following code:

```
from fastapi import FastAPI
from fastapi.responses import RedirectResponse
from database.connection import conn

from routes.users import user_router
from routes.events import event_router

import uvicorn

app = FastAPI()

# Register routes

app.include_router(user_router,  prefix="/user")
app.include_router(event_router, prefix="/event")

@app.on_event("startup")
def on_startup():
    conn()

@app.get("/")
async def home():
    return RedirectResponse(url="/event/")

if __name__ == '__main__':
    uvicorn.run("main:app", host="0.0.0.0", port=8080,
    reload=True)
```

The database will be created once the application starts. In the startup event, we have called the `conn()` function responsible for creating the database. Start the application in your terminal and you should see the output in your console, indicating that the database has been created as well as the table:



```
INFO:     Will watch for changes in these directories: ['/Users/youngestdev/Work/Building-Web-APIs-with-FastAPI-a
nd-Python/ch06/planner']
INFO:     Uvicorn running on http://0.0.0.0:8080 (Press CTRL+C to quit)
INFO:     Started reloader process [5668] using statreload
INFO:     Started server process [5671]
INFO:     Waiting for application startup.
2022-04-06 16:25:36,230 INFO sqlalchemy.engine.Engine BEGIN (implicit)
2022-04-06 16:25:36,230 INFO sqlalchemy.engine.Engine PRAGMA main.table_info("event")
2022-04-06 16:25:36,230 INFO sqlalchemy.engine.Engine [raw sql] ()
2022-04-06 16:25:36,230 INFO sqlalchemy.engine.Engine PRAGMA temp.table_info("event")
2022-04-06 16:25:36,231 INFO sqlalchemy.engine.Engine [raw sql] ()
2022-04-06 16:25:36,231 INFO sqlalchemy.engine.Engine
CREATE TABLE event (
        tags JSON,
        id INTEGER,
        title VARCHAR NOT NULL,
        image VARCHAR NOT NULL,
        description VARCHAR NOT NULL,
        location VARCHAR NOT NULL,
        PRIMARY KEY (id)
)


2022-04-06 16:25:36,231 INFO sqlalchemy.engine.Engine [no key 0.00003s] ()
2022-04-06 16:25:36,232 INFO sqlalchemy.engine.Engine COMMIT
```

Figure 6.1 – The planner database and event table created successfully

The SQL commands displayed in the terminal are there because of setting `echo` to `True` when creating the database engine. Now that we have successfully created the database, let's update our events' CRUD operation routes to use the database.

Creating events

Let's look at the steps:

1. In `routes/events.py`, update the imports to include the Event table model class as well as the `get_session()` function. The `get_session()` function is imported so that the routes can access the session object created:

```
from fastapi import APIRouter, Depends, HTTPException,
Request, status
from database.connection import get_session
from models.events import Event, EventUpdate
```

What Is Depends?

The `Depends` class is responsible for exercising dependency injection in FastAPI applications. The `Depends` class takes a truth source such as a function as an argument and is passed as a function argument in a route, mandating that the dependency condition be satisfied before any operation can be executed.

2. Next, let's update the `POST` route function responsible for creating a new event, `create_event()`:

```
@event_router.post("/new")
async def create_event(new_event: Event,
session=Depends(get_session)) -> dict:
    session.add(new_event)
    session.commit()
    session.refresh(new_event)

    return {
        "message": "Event created successfully"
    }
```

   In this code block, we have indicated that the session object required to execute database transactions is dependent on the `get_session()` function we created earlier.

   In the function body, the data is added to the session and then committed to the database, after which the database is refreshed.

3. Let's test the routes to preview changes:

```
(venv)$ curl -X 'POST' \
  'http://0.0.0.0:8080/event/new' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "title": "FastAPI Book Launch",
  "image": "fastapi-book.jpeg",
  "description": "We will be discussing the contents
  of the FastAPI book in this event. Ensure to come
  with your own copy to win gifts!",
```