Async SQL (Relational) Databases

You can also use `encode/databases` with **FastAPI** to connect to databases using `async` and `await` .

It is compatible with:

- PostgreSQL
- MySQL
- SQLite

In this example, we'll use **SQLite**, because it uses a single file and Python has integrated support. So, you can copy this example and run it as is.

Later, for your production application, you might want to use a database server like **PostgreSQL**.

!!! tip You could adopt ideas from the section about SQLAlchemy ORM (SQL (Relational) Databases{.internal-link target=_blank}), like using utility functions to perform operations in the database, independent of your **FastAPI** code.

```
This section doesn't apply those ideas, to be equivalent to the counterpart in <a
href="https://www.starlette.io/database/" class="external-link"
target="_blank">Starlette</a>.
```

Import and set up `SQLAlchemy`

- Import `SQLAlchemy` .
- Create a `metadata` object.
- Create a table `notes` using the `metadata` object.

```
{!../../../docs_src/async_sql_databases/tutorial001.py!}
```

!!! tip Notice that all this code is pure SQLAlchemy Core.

```
`databases` is not doing anything here yet.
```

Import and set up `databases`

- Import `databases` .
- Create a `DATABASE_URL` .
- Create a `database` object.

```
{!../../../docs_src/async_sql_databases/tutorial001.py!}
```

!!! tip If you were connecting to a different database (e.g. PostgreSQL), you would need to change the `DATABASE_URL` .

Create the tables

In this case, we are creating the tables in the same Python file, but in production, you would probably want to create them with Alembic, integrated with migrations, etc.

Here, this section would run directly, right before starting your **FastAPI** application.

- Create an `engine`.
- Create all the tables from the `metadata` object.

```
{!../../../docs_src/async_sql_databases/tutorial001.py!}
```

Create models

Create Pydantic models for:

- Notes to be created (`NoteIn`).
- Notes to be returned (`Note`).

```
{!../../../docs_src/async_sql_databases/tutorial001.py!}
```

By creating these Pydantic models, the input data will be validated, serialized (converted), and annotated (documented).

So, you will be able to see it all in the interactive API docs.

Connect and disconnect

- Create your `FastAPI` application.
- Create event handlers to connect and disconnect from the database.

```
{!../../../docs_src/async_sql_databases/tutorial001.py!}
```

Read notes

Create the *path operation function* to read notes:

```
{!../../../docs_src/async_sql_databases/tutorial001.py!}
```

!!! Note Notice that as we communicate with the database using `await`, the *path operation function* is declared with `async`.

Notice the `response_model=List[Note]`

It uses `typing.List`.

That documents (and validates, serializes, filters) the output data, as a `list` of `Note`s.

Create notes

Create the *path operation function* to create notes:

```
{!../../../docs_src/async_sql_databases/tutorial001.py!}
```

!!! Note Notice that as we communicate with the database using `await`, the *path operation function* is declared with `async`.

About `{**note.dict(), "id": last_record_id}`

`note` is a Pydantic `Note` object.

`note.dict()` returns a `dict` with its data, something like:

```
{
    "text": "Some note",
    "completed": False,
}
```

but it doesn't have the `id` field.

So we create a new `dict`, that contains the key-value pairs from `note.dict()` with:

```
{**note.dict()}
```

`**note.dict()` "unpacks" the key value pairs directly, so, `{**note.dict()}` would be, more or less, a copy of `note.dict()`.

And then, we extend that copy `dict`, adding another key-value pair: `"id": last_record_id`:

```
{**note.dict(), "id": last_record_id}
```

So, the final result returned would be something like:

```
{
    "id": 1,
    "text": "Some note",
    "completed": False,
}
```

Check it

You can copy this code as is, and see the docs at http://127.0.0.1:8000/docs.

There you can see all your API documented and interact with it:

More info

You can read more about [`encode/databases` at its GitHub page](https://github.com/encode/databases).