10:30:18

- Sqlalchemy does not tracks the changes made to database table schemas, neither applies the changes automatically to the existing tables.

- To automatically update the database table schemas, based on changes in model classes, we use ALEMBIC.

ALEMBIC :-

- Alembic is a Database Migration Tool.
- Able to track incremental changes in database

Setting Up :-

10:32:40

- Database migrations :- It allows us to incrementally track changes to our database (i.e, each change is stored as a checkpoint, and we can rollback to previous checkpoints, like Git).

- Alembic can also automatically pull database models from sqlalchemy and generate proper tables.

- Installing alembic :-

```
pip install alembic
```

⇒ Cmd command to get alembic help :-

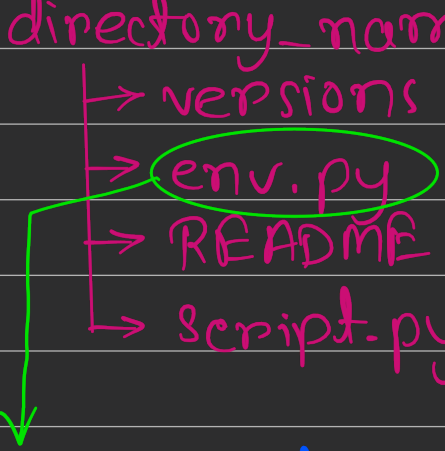
```
alembic --help
```

(1) At first, we need to initialise alembic.
This creates an alembic directory.
Command for initialising alembic :-

```
alembic init directory_name
```

⇒ It will create a folder named directory_name outside the project directory, where generally the venv folder and .env file exists.
Also, at that place, it will create a file with name directory_name and .ini extension.

(2) The directory_name folder contains following :-



The env.py is the main Configuration File.

(3) As alembic works with sqlalchemy models, thus, it needs access to the Base object, made in the database.py file, that is imported

to the models.py file, as follows :-

```
from .database import Base
```

We import this Base object in the env.py file of alembic directory :-

```
from crud_api_project_orm.models import Base
```

(crud_api_project_orm → project name)

must be imported from models & not from database

with this object, we can access all the created sqlalchemy models.

→ Then, assign this object to the target_metadata variable in that file (replace the None) :-

```
target_metadata = Base.metadata
```

---

(4) We need to update the .ini file created by alembic.

We need to assign value to sqlalchemy.url

→ driver will be postgresql+psycopg2

user → username

...

For our case, the url will be :-

```
sqlalchemy.url = postgresql+psycopg2://
...postgres:Ultimate99@localhost/fastapi_orm
```

:port → optional

> But, instead of hardcoding the values of hostname, username, password, we will override the value of sqlalchemy.url stored in directory_name.ini file (for our case, it is alembic.ini file), within the alembic env.py file.

To override the value in env.py file, we need to code as follow :-

```
config = context.config
config.set_main_option(
    "sqlalchemy.url", "driver://user:pass@localhost/dbname"
)
```

config = context.config — this exists already. Below it, we insert our code

with this function, we set the value to variables of alembic.ini file.

1st argument is the name of the variable, whose value we want to set

2nd parameter is value that we want to assign to the variable

> We are setting the value of sqlalchemy.url variable from here, because, we want the credentials to be imported from .env file (and not hardcoded). To import & use them, we need python code, & thus, we are overriding it in env.py [a python script], and then, will use environment variables there.

⇒ At this point, Alembic is set to connect to our database, & modify the tables, and make operations.

10:43:45

⇒ Now, we will delete all the tables in our database, and begin building them from scratch, using Alembic

⇒ Working with Alembic :-

⇒ We type alembic commands in cmd.

1) [alembic --help
Shows all the commands.

2) When we want to make changes in our database, we make a revision. It tracks the changes in our database, step-by-step [exactly similar to git commit command function] :-

[alembic revision --help
Shows all the flags, available to be used with the revision command.

[alembic revision -m "Create Post Table"
adds documentation message specific to the revision

As soon as the revision command is executed, we can find a python file being created in the versions folder inside alembic directory.

This file contains all the details of changes in the revision.

This revision file contains

(a) revision id in the revision variable.

(b) upgrade() function :-

This function contains the commands, that will make the changes, to be made in this revision

Ex:- For a revision that is to create post table, we will keep all the code to create the post table inside this upgrade() function.

(c) downgrade() function :-

This function will contain the commands, that will rollback the changes made in the upgrade.

Ex :- For revision that creates posts table, code to drop the posts table will be in the downgrade() function.

Both the process of revision updation and rollback is manual, and not automatic as in git.

10:46:00

Sample of upgrade() & downgrade() function definition is as follow :-

```
def upgrade():
    op.create_table('posts', sa.Column('id', sa.Integer(), nullable=False,
                    primary_key=True), sa.Column('title', sa.String(), nullable=False))
    pass

def downgrade():
    op.drop_table('posts')
    pass
```

table name — column 1 — column name — sqlalchemy — alembic — column 2

Now, we can set database to upgrade to this point, using the upgrade command :-

`alembic upgrade --help`

Shows help of what flags this command needs.

- It takes the revesion id/number of the revision to which we want to upgrade.
- This id can be found in the revision variable of the python file, inside the versions folder.

Ex:- say, the id in file is revision = "ccff4d0"

Then, command to upgrade :-

`alembic upgrade ccff4d0` (without quotes in the command)

(or) `alembic upgrade (+1)` → will upgrade revision by 1

It will create the table as stated in the upgrade() function definition.

It will also create a table named alembic_version which will contain 1 column, named version_num where all the revision ids will get stored.

To know at which migration version/revision our database is at currently, we use following command :-

`alembic current`

Now, to make changes in database, we create new revision, and the code for new changes should be done in that revisions

upgrade() function.

Ex:- Making revision for adding new column to posts table

`alembic revision -m "add content column to ... posts table"`

The upgrade() & downgrade() function definitions for the new revision :-

```
# revision identifiers, used by Alembic.
revision = '01b2584928a5'
down_revision = 'cfcc4fd02d18'
branch_labels = None
depends_on = None


def upgrade():
    op.add_column('posts', sa.Column('content', sa.String(), nullable=False))
    pass


def downgrade():
    op.drop_column('posts', 'content')
    pass
```

for 2nd revision onward, every revision will have down_revision id, same previous revision id

function to add new column

column datatype

table name in which we are adding the column

new column name

constraint

as in upgrade() we are adding a column thus in downgrade() we are dropping that column (completely opposite)

The latest revision is called, **head**.
We can find out the latest/last created revision id, by the following command :-

`alembic heads`

To upgrade to the latest revision, without knowing the revision, command is :-

`alembic upgrade head` → refers to the revision id of latest revision

07 Command to downgrade database revision :-

`alembic downgrade ccff10d4`

⇒ this will downgrade revision to the mentioned revision id :-

⇒ Without knowing revision id, if we want to downgrade, it can be done as :-

`alembic downgrade -1`

will rollback to 1 revision earlier

`alembic downgrade -2`

will rollback to 2 revisions earlier.

⇒ Adding new table will also be using new revision :-

`alembic revision -m "add users table"`

upgrade() & downgrade() of this revision :-

```python
def upgrade():
    op.create_table('users',
                    sa.Column('id', sa.Integer(), nullable=False),
                    sa.Column('email', sa.String(), nullable=False),
                    sa.Column('password', sa.String(), nullable=False),
                    sa.Column('created_at', sa.TIMESTAMP(timezone=True),
                              server_default=sa.text('now()'), nullable=False),
                    sa.PrimaryKeyConstraint('id'),
                    sa.UniqueConstraint('email')
                    )
    pass


def downgrade():
    op.drop_table("users")
```

Adding owner_id column to posts table, and setting it as foreign key :-

- [alembic revision -m "add foreign-key to posts"]

- Function definitions :-

```
def upgrade():
    op.add_column('posts', sa.Column('owner_id', sa.Integer(), nullable=False))
    op.create_foreign_key('post_users_fk', source_table="posts", referent_table="users",
    local_cols=[...
                ...'owner_id'], remote_cols=['id'], ondelete="CASCADE")
    pass


def downgrade():
    op.drop_constraint('post_users_fk', table_name="posts")
    op.drop_column('posts', 'owner_id')
    pass
```

table name to which foreign key is referencing

constraint name

table name from which constraint is to be dropped.

table name in which column to be dropped exists.

column name to be dropped

- Apply the revision :-

  [alembic revision head]

11:05:25

- Add remaining columns to posts table.