SQLAlchemy: engine, connection and session difference

Asked 8 years, 2 months ago Modified 1 month ago Viewed 174k times

283

I use SQLAlchemy and there are at least three entities: `engine`, `session` and `connection`, which have `execute` method, so if I e.g. want to select all records from `table` I can do this on the **Engine level**:

```
engine.execute(select([table])).fetchall()
```

and on the **Connection level**:

```
connection.execute(select([table])).fetchall()
```

and even on the **Session level**:

```
session.execute(select([table])).fetchall()
```

- the results will be the same.

As I understand it, if someone uses `engine.execute` it creates `connection`, opens `session` (Alchemy takes care of it for you) and executes the query. But is there a global difference between these three ways of performing such a task?

python session orm sqlalchemy psycopg2

Share Edit Follow Flag

edited Jan 5 at 13:24

Konstantin A. Magg
1,118 9 20

asked Dec 16, 2015 at 21:29

ololobus
3,758 2 19 23

3 Answers

Sorted by: Highest score (default)

276

Running .execute()

When executing a plain `SELECT * FROM tablename`, there's no difference in the result provided.

The differences between these three objects do become important depending on the context that the `SELECT` statement is used in or, more commonly, when you want to do

other things like `INSERT`, `DELETE`, etc.

When to use Engine, Connection, Session generally

- **Engine** is the lowest level object used by SQLAlchemy. It maintains a pool of connections available for use whenever the application needs to talk to the database. `.execute()` is a convenience method that first calls `conn = engine.connect(close_with_result=True)` and the then `conn.execute()`. The close_with_result parameter means the connection is closed automatically. (I'm slightly paraphrasing the source code, but essentially true). *edit: Here's the source code for engine.execute*

  You can use engine to execute raw SQL.

  ```
  result = engine.execute('SELECT * FROM tablename;')
  # what engine.execute() is doing under the hood:
  conn = engine.connect(close_with_result=True)
  result = conn.execute('SELECT * FROM tablename;')

  # after you iterate over the results, the result and connection get closed
  for row in result:
      print(result['columnname']

  # or you can explicitly close the result, which also closes the connection
  result.close()
  ```

  This is covered in the docs under basic usage.

- **Connection** is (as we saw above) the thing that actually does the work of executing a SQL query. You should do this whenever you want greater control over attributes of the connection, when it gets closed, etc. An important example of this is a transaction, which lets you decide when to commit your changes to the database (if at all). In normal use, changes are auto-committed. With the use of transactions, you could (for example) run several different SQL statements and if something goes wrong with one of them you could undo all the changes at once.

  ```
  connection = engine.connect()
  trans = connection.begin()
  try:
      connection.execute(text("INSERT INTO films VALUES ('Comedy', '82
minutes');"))
      connection.execute(text("INSERT INTO datalog VALUES ('added a
comedy');"))
      trans.commit()
  except Exception:
      trans.rollback()
      raise
  ```

  This would let you undo both changes if one failed, like if you forgot to create the datalog table.

  So if you're executing raw SQL code and need control, use connections

- **Sessions** are used for the Object Relationship Management (ORM) aspect of

SQLAlchemy (in fact you can see this from how they're imported: `from sqlalchemy.orm import sessionmaker`). They use connections and transactions under the hood to run their automatically-generated SQL statements. `.execute()` is a convenience function that passes through to whatever the session is bound to (usually an engine, but can be a connection).

If you're using the ORM functionality, use a session. If you're only doing straight SQL queries not bound to objects, you're probably better off using connections directly.

Share Edit Follow Flag

edited Apr 11, 2023 at 20:34

Konstantin Smolyanin
18.1k 12 60 57

answered Mar 13, 2017 at 20:16

Neal
3,262 1 14 16

Given created session, how does my Session linked with my PostgreSQL connection ? – Jeff Bootsholz Jun 3, 2019 at 14:55

1 @RajuyourPepe `my_session.connection()` . Docs: docs.sqlalchemy.org/en/13/orm/... – Neal Jun 3, 2019 at 18:55

Seriously ? 'Session' object has no attribute 'connect'", is what I have found – Jeff Bootsholz Jun 4, 2019 at 3:36

1 @RajuyourPepe look carefully at the command, it is `connection()` not `connect` . See my link to the docs. – Neal Jun 4, 2019 at 13:52

2 I am using session and I also close it when querying is done. I still sometimes get that database is locked. Any idea? – Yash Tamakuwala Nov 4, 2020 at 7:03

199

A one-line overview:

The behavior of `execute()` is same in all the cases, but they are 3 different methods, in `Engine` , `Connection` , and `Session` classes.

What exactly is `execute()` :

To understand behavior of `execute()` we need to look into the `Executable` class. `Executable` is a superclass for all "statement" types of objects, including select(), delete(),update(), insert(), text() - in simplest words possible, an `Executable` is a SQL expression construct supported in SQLAlchemy.

In all the cases the `execute()` method takes the SQL text or constructed SQL expression i.e. any of the variety of SQL expression constructs supported in SQLAlchemy and returns query results (a `ResultProxy` - Wraps a `DB-API` cursor object to provide easier access to row columns.)

To clarify it further (only for conceptual clarification, not a recommended

approach):

In addition to `Engine.execute()` (connectionless execution), `Connection.execute()`, and `Session.execute()`, it is also possible to use the `execute()` directly on any `Executable` construct. The `Executable` class has it's own implementation of `execute()` - As per official documentation, one line description about what the `execute()` does is "**Compile and execute this `Executable`**". In this case we need to explicitly bind the `Executable` (SQL expression construct) with a `Connection` object or, `Engine` object (which implicitly get a `Connection` object), so the `execute()` will know where to execute the `SQL`.

The following example demonstrates it well - Given a table as below:

```
from sqlalchemy import MetaData, Table, Column, Integer

meta = MetaData()
users_table = Table('users', meta,
    Column('id', Integer, primary_key=True),
    Column('name', String(50)))
```

Explicit execution i.e. `Connection.execute()` - passing the SQL text or constructed SQL expression to the `execute()` method of `Connection`:

```
engine = create_engine('sqlite:///file.db')
connection = engine.connect()
result = connection.execute(users_table.select())
for row in result:
    # ....
connection.close()
```

Explicit connectionless execution i.e. `Engine.execute()` - passing the SQL text or constructed SQL expression directly to the `execute()` method of Engine:

```
engine = create_engine('sqlite:///file.db')
result = engine.execute(users_table.select())
for row in result:
    # ....
result.close()
```

Implicit execution i.e. `Executable.execute()` - is also connectionless, and calls the `execute()` method of the `Executable`, that is, it calls `execute()` method directly on the `SQL` expression construct (an instance of `Executable`) itself.

```
engine = create_engine('sqlite:///file.db')
meta.bind = engine
result = users_table.select().execute()
for row in result:
    # ....
result.close()
```

Note: Stated the implicit execution example for the purpose of clarification - this way of execution is highly not recommended - as per docs:

> "implicit execution" is a very old usage pattern that in most cases is more confusing than it is helpful, and its usage is discouraged. Both patterns seem to encourage the overuse of expedient "short cuts" in application design which lead to problems later on.

Your questions:

> As I understand if someone use engine.execute it creates connection, opens session (Alchemy cares about it for you) and executes query.

You're right for the part "if someone use `engine.execute` it creates `connection` " but not for "opens `session` (Alchemy cares about it for you) and executes query " - Using `Engine.execute()` and `Connection.execute()` is (almost) one the same thing, in formal, `Connection` object gets created implicitly, and in later case we explicitly instantiate it. What really happens in this case is:

```
`Engine` object (instantiated via `create_engine()`) -> `Connection` object
(instantiated via `engine_instance.connect()`) -> `connection.execute({*SQL
expression*})`
```

> But is there a global difference between these three ways of performing such task?

At DB layer it's exactly the same thing, all of them are executing SQL (text expression or various SQL expression constructs). From application's point of view there are two options:

- Direct execution - Using `Engine.execute()` or `Connection.execute()`
- Using `sessions` - efficiently handles transaction as single unit-of-work, with ease via `session.add()`, `session.rollback()`, `session.commit()`, `session.close()`. It is the way to interact with the DB in case of ORM i.e. mapped tables. Provides identity_map for instantly getting already accessed or newly created/added objects during a single request.

`Session.execute()` ultimately uses `Connection.execute()` statement execution method in order to execute the SQL statement. Using `Session` object is SQLAlchemy ORM's recommended way for an application to interact with the database.

An excerpt from the docs:

Its important to note that when using the SQLAlchemy ORM, these objects are not generally accessed; instead, the Session object is used as the interface to the database. However, for applications that are built around direct usage of textual SQL statements and/or SQL expression constructs without involvement by the ORM's higher level management services, the Engine and Connection are king (and queen?) - read on.

Share Edit Follow Flag

edited Nov 1, 2017 at 2:06
Zak
963 10 20

answered Dec 18, 2015 at 21:32
Nabeel Ahmed
18.7k 4 58 63

1 "connectionless" word implies no connection is being created, which according to Neal's answer is not the case. – Atom Mar 7, 2019 at 10:26

With `Connection` you can also use `commit()` and `rollback()`, so I wonder when does it make sense to use `Session` instead? The benefit of `Connection` is that it is `Connectable` and because of it, it can be used easier with pandas I think. – Karol Zlot Dec 7, 2022 at 2:49

Executable.execute() is deprecated since 1.4 and will be removed in 2.0. All statement execution in 2.0 is performed by the Connection.execute() or by the Session.execute() docs.sqlalchemy.org/en/14/core/... – hashlash Jan 18, 2023 at 5:28

0

Here is an example of running DCL (Data Control Language) such as GRANT

```
def grantAccess(db, tb, user):
  import sqlalchemy as SA
  import psycopg2

  url = "{d}+{driver}://{u}:{p}@{h}:{port}/{db}".\
            format(d="redshift",
            driver='psycopg2',
            u=username,
            p=password,
            h=host,
            port=port,
            db=db)
  engine = SA.create_engine(url)
  cnn = engine.connect()
  trans = cnn.begin()
  strSQL = "GRANT SELECT on table " + tb + " to " + user + " ;"
  try:
      cnn.execute(strSQL)
      trans.commit()
  except:
      trans.rollback()
      raise
```

Share Edit Follow Flag

answered Aug 12, 2019 at 21:31

Jie

1,176 1 15 19

you didn't check whether the db is alive or not? – greendino Oct 16, 2020 at 5:24