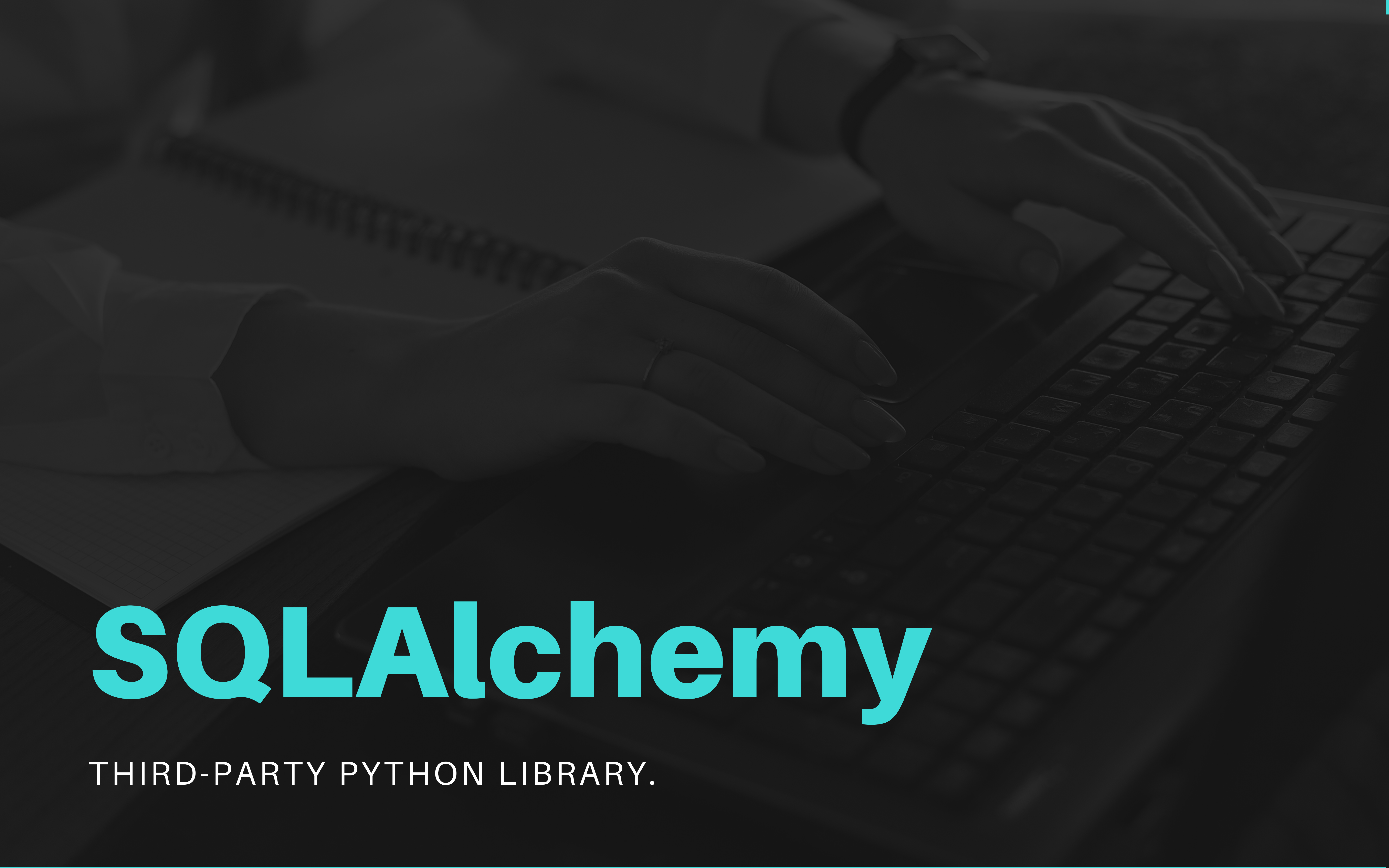


SQLAlchemy

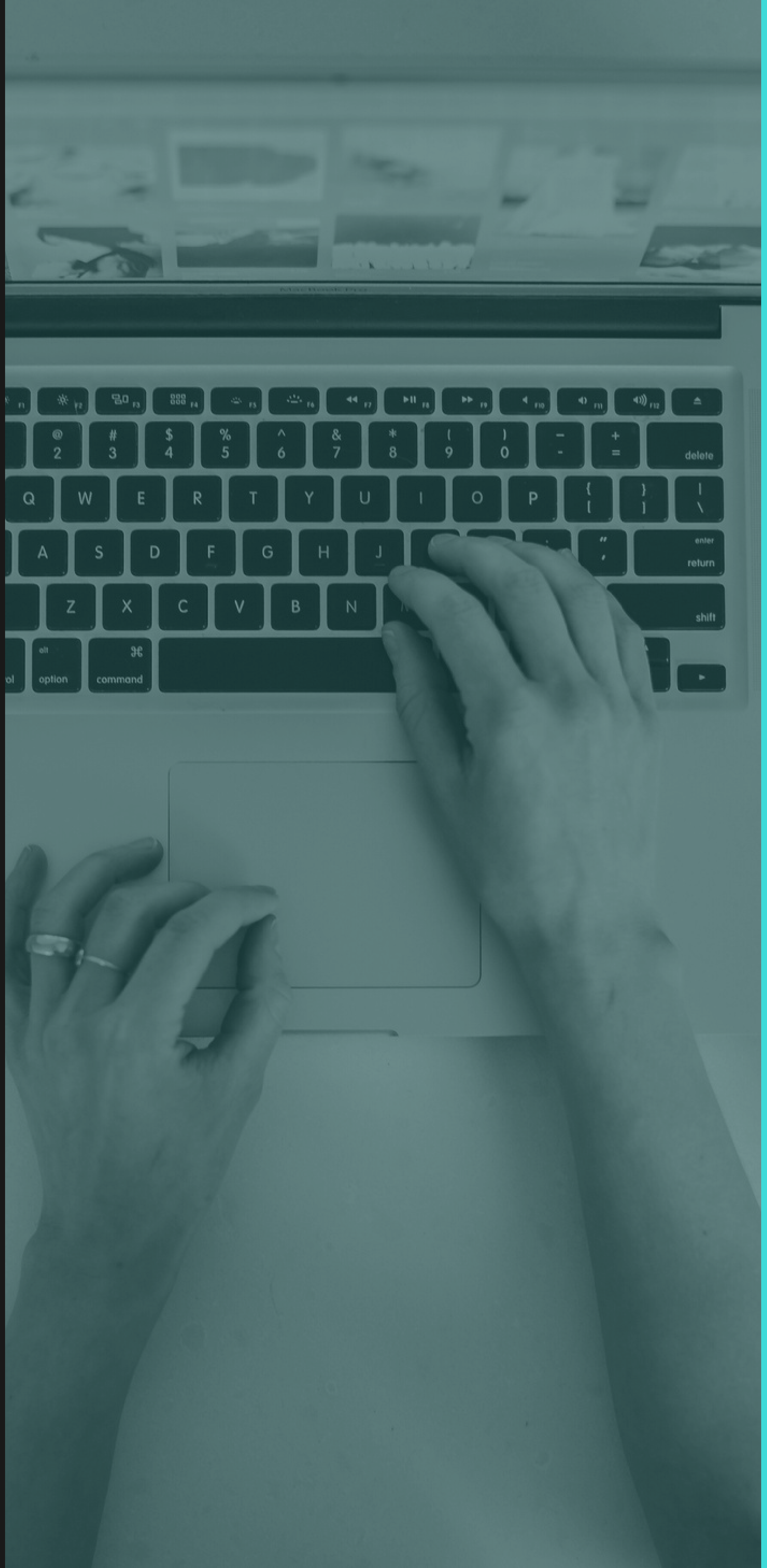
THIRD-PARTY PYTHON LIBRARY.

ADVANCED PYTHON PROGRAMMING

What is SQLAlchemy

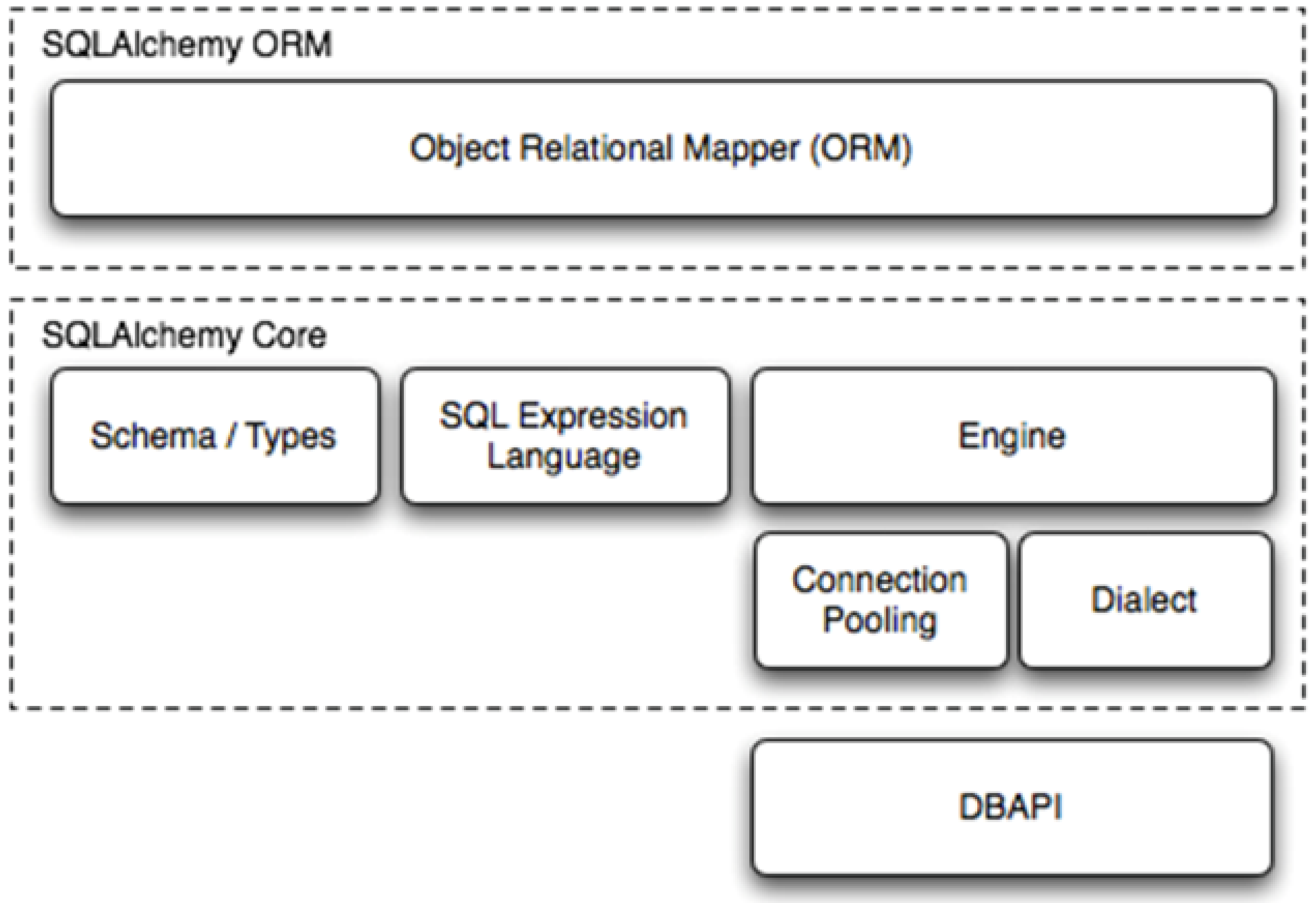
SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives developers the full power and flexibility of SQL.

It is a comprehensive set of tools for working with databases and Python. It has several distinct areas of functionality which can be used individually or combined together



Major Components

...with Component Dependencies



SQLAlchemy Core

SQLAlchemy Core is the foundational component of the library that provides developers with a low-level interface for interacting with databases using Python.

Core allows developers to work with databases at a granular level, offering fine-grained control over database interactions while maintaining a Pythonic programming interface.

It provides tools for managing connectivity to a database, interacting with database queries and results, and programmatic construction of SQL statements.

CORE DEPENDENCIES

Schema

The core of SQLAlchemy’s query and object mapping operations are supported by database metadata, which is comprised of Python objects that describe tables and other schema-level objects.

SQL Expression Language

It is a Pythonic interface for generating SQL queries and expressions dynamically. It provides a set of Python classes, functions, and operators that map to SQL constructs, allowing developers to construct SQL queries using Python code.

CORE DEPENDENCIES -- ENGINE

The start of any SQLAlchemy application is an object called the Engine. This object acts as a central source of connections to a particular database. The engine is typically a global object created just once for a particular database server, and is configured using a URL string which will describe how it should connect to the database host or backend.

Connection Pooling

Standard technique used to maintain long running connections in memory for efficient re-use, as well as to provide management for the total number of connections an application might use simultaneously

Dialects

Python object that acts as a special language translator, helping the library communicate with different types of databases, such as SQLite, PostgreSQL, or MySQL.

DBAPI

Third party driver that SQLAlchemy uses to interact with a particular database.

SQLAlchemy ORM

The Object Relational Mapper (ORM) builds upon the Core to provide optional object relational mapping capabilities.

SQLAlchemy ORM works with the Core to make it easier to use. With ORM's additional configuration layer, one can connect Python classes to database tables and use a feature called the Session to save and load objects from the database.

Through the Core-level SQL Expressions Language, it allows the writing of SQL queries using Python objects instead of writing out the actual or more precise SQL statements directly.

While both Core and ORM components offer database interaction capabilities, they serve different purposes and cater to different use cases.

Core highlights **control over ease and flexibility**
Whereas ORM highlights **ease and flexibility over control**

CORE

- **Low-Level Database Interaction:** The Core component provides a low-level interface for interacting directly with databases. It includes features for managing database connections, executing SQL queries, and handling transactions.
- **SQL Expression Language**: Core offers a SQL Expression Language for constructing SQL queries and commands dynamically using Python syntax and operators.
- **Database Metadata Handling**: Core provides tools for reflecting database metadata, such as table structures, and column types, to access and manipulate schema information.
- **Dialect Handling**: Core includes database-specific dialects that handle differences between various database systems.

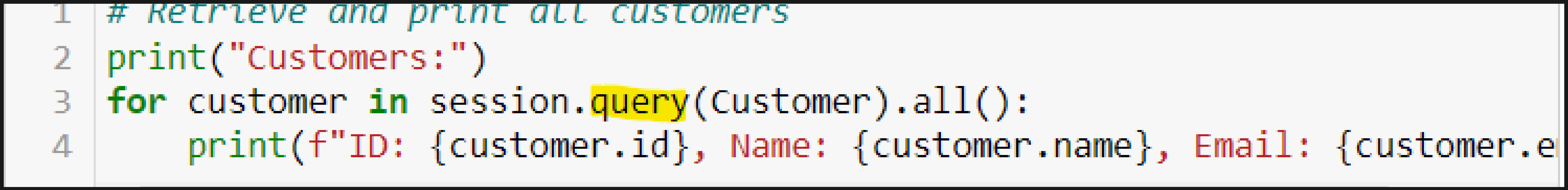
ORM

- **Object-Relational Mapping (ORM):** The ORM component provides high-level object-relational mapping capabilities. It allows mapping Python classes to database tables.
- **Unit of Work Pattern:** The ORM automates the task of persisting objects to the database using a pattern called the unit of work. It tracks changes made to objects and translates them into corresponding SQL statements.
- **Session Management:** ORM introduces the concept of a Session, which serves as a central hub for managing interactions with the database. It handles object identity tracking, and transaction management.
- **Object-Centric Querying**: ORM extends the Core's SQL Expression Language to support object-centric querying, using Python objects and methods.

Functionality of SQLAlchemy

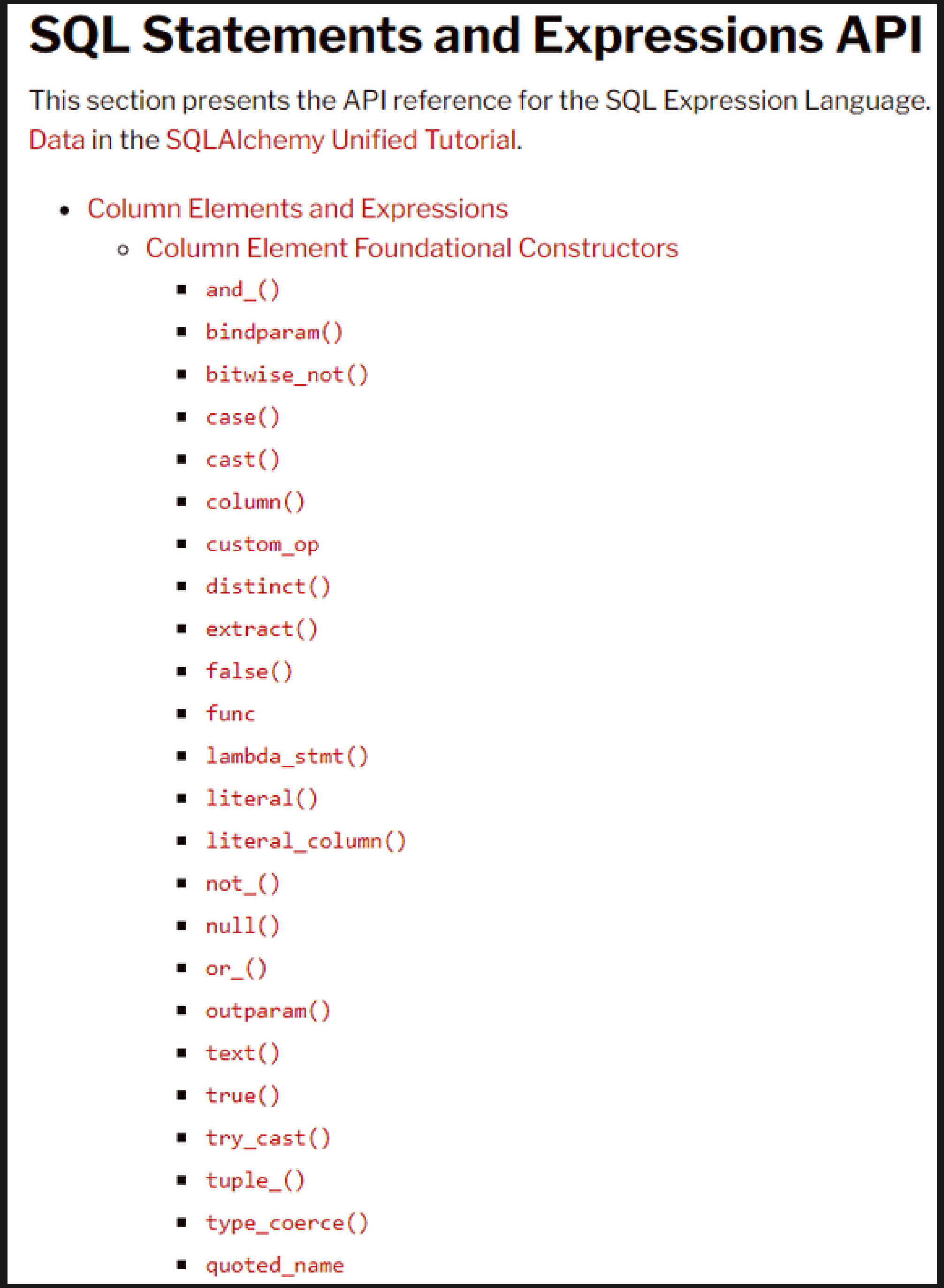
SQL Expression Language

SQLAlchemy provides a powerful SQL Expression Language that allows you to construct SQL queries using Pythonic syntax. This language enables you to generate SQL statements dynamically, making it easier to interact with databases programmatically.



```python
# Retrieve and print all customers
print("Customers:")
for customer in session.query(Customer).all():
    print(f"ID: {customer.id}, Name: {customer.name}, Email: {customer.e
```

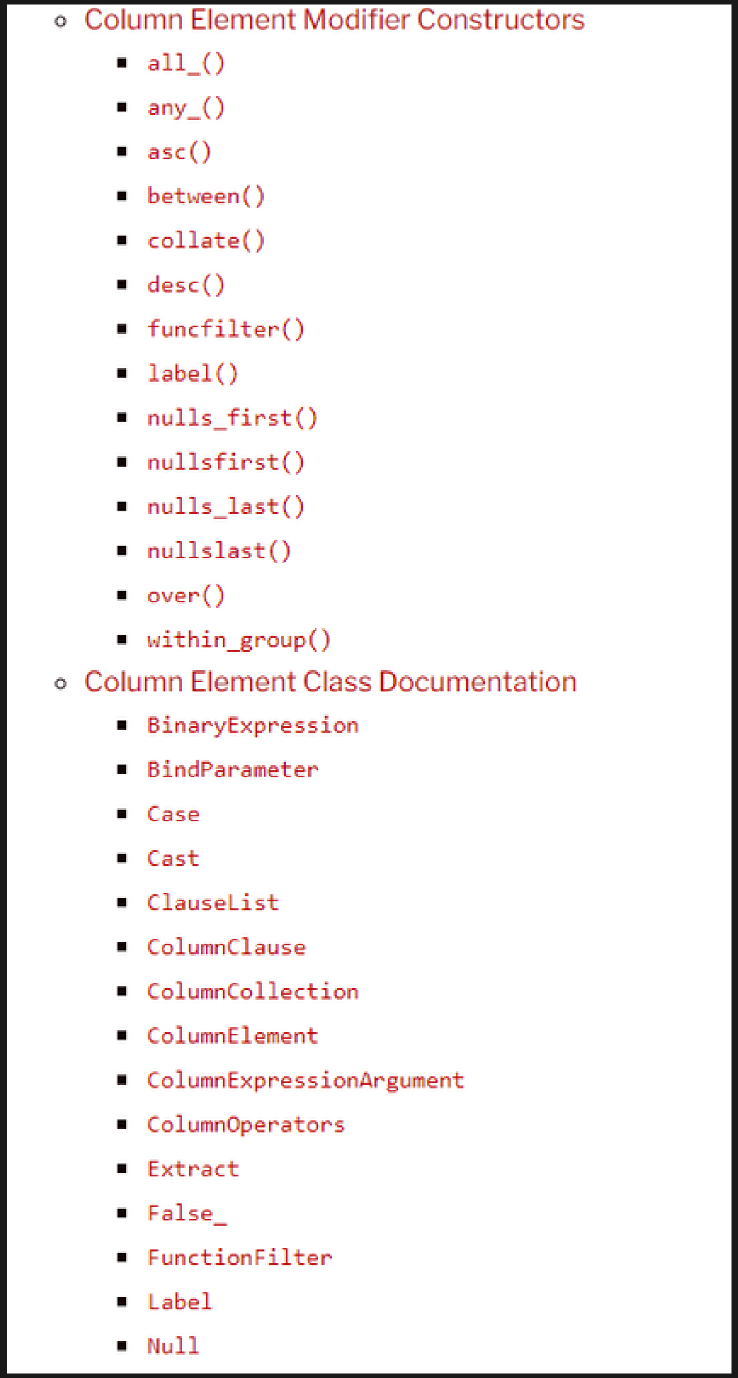
SQL Expression Language



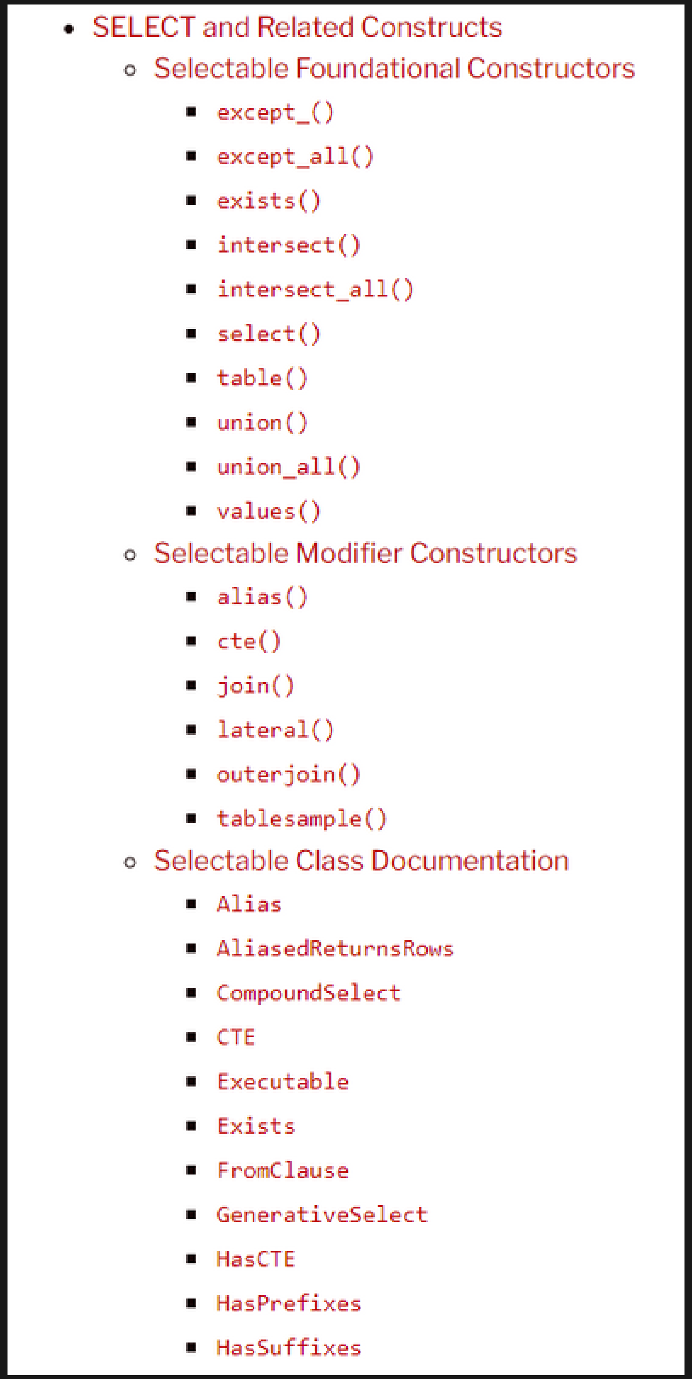
SQL Statements and Expressions API

This section presents the API reference for the SQL Expression Language.
Data in the SQLAlchemy Unified Tutorial.

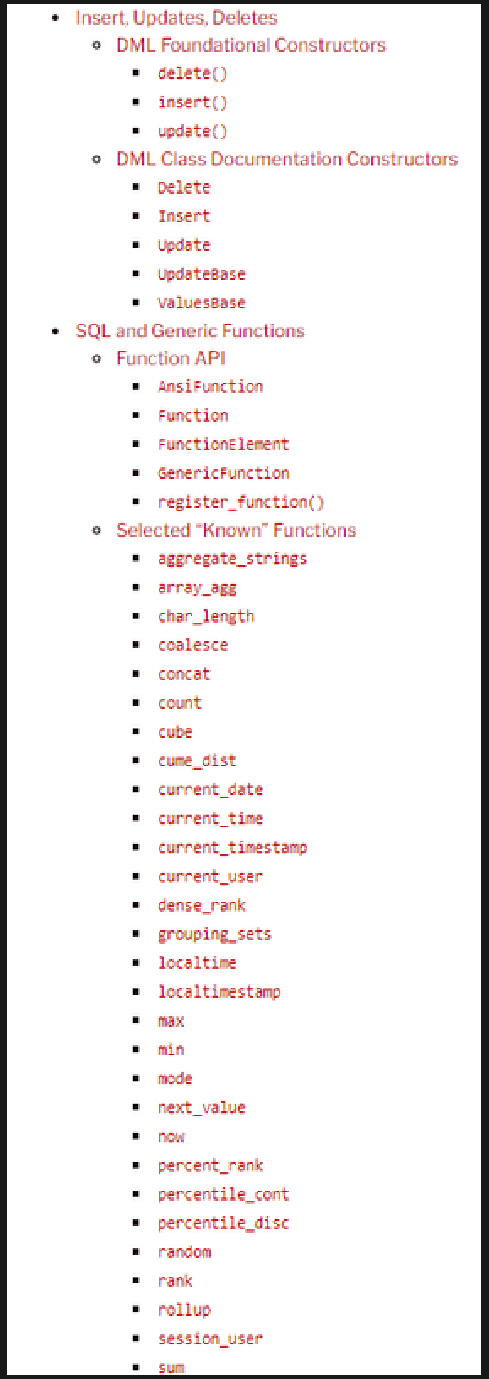
- Column Elements and Expressions
  - Column Element Foundational Constructors
    - `and_()`
    - `bindparam()`
    - `bitwise_not()`
    - `case()`
    - `cast()`
    - `column()`
    - `custom_op`
    - `distinct()`
    - `extract()`
    - `false()`
    - `func`
    - `lambda_stmt()`
    - `literal()`
    - `literal_column()`
    - `not_()`
    - `null()`
    - `or_()`
    - `outparam()`
    - `text()`
    - `true()`
    - `try_cast()`
    - `tuple_()`
    - `type_coerce()`
    - `quoted_name`



- Column Element Modifier Constructors
  - `all_()`
  - `any_()`
  - `asc()`
  - `between()`
  - `collate()`
  - `desc()`
  - `funcfilter()`
  - `label()`
  - `nulls_first()`
  - `nullsfirst()`
  - `nulls_last()`
  - `nullslast()`
  - `over()`
  - `within_group()`
- Column Element Class Documentation
  - `BinaryExpression`
  - `BindParameter`
  - `Case`
  - `Cast`
  - `ClauseList`
  - `ColumnClause`
  - `ColumnCollection`
  - `ColumnElement`
  - `ColumnExpressionArgument`
  - `ColumnOperators`
  - `Extract`
  - `False_`
  - `FunctionFilter`
  - `Label`
  - `Null`



- SELECT and Related Constructs
  - Selectable Foundational Constructors
    - `except_()`
    - `except_all()`
    - `exists()`
    - `intersect()`
    - `intersect_all()`
    - `select()`
    - `table()`
    - `union()`
    - `union_all()`
    - `values()`
  - Selectable Modifier Constructors
    - `alias()`
    - `cte()`
    - `join()`
    - `lateral()`
    - `outerjoin()`
    - `tablesample()`
  - Selectable Class Documentation
    - `Alias`
    - `AliasedReturnsRows`
    - `CompoundSelect`
    - `CTE`
    - `Executable`
    - `Exists`
    - `FromClause`
    - `GenerativeSelect`
    - `HasCTE`
    - `HasPrefixes`
    - `HasSuffixes`

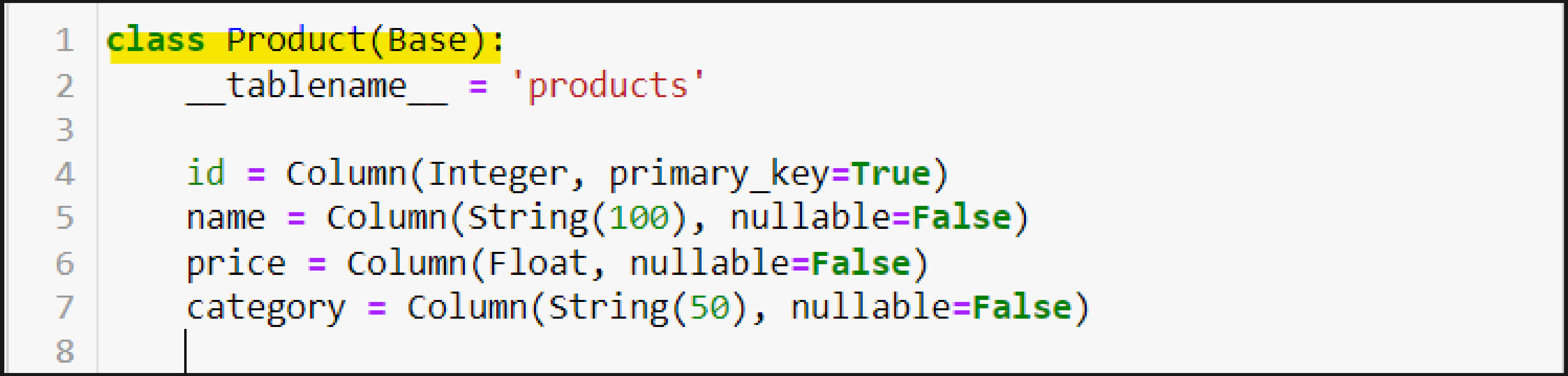


- Insert, Updates, Deletes
  - DML Foundational Constructors
    - `delete()`
    - `insert()`
    - `update()`
  - DML Class Documentation Constructors
    - `Delete`
    - `Insert`
    - `Update`
    - `UpdateBase`
    - `ValuesBase`
- SQL and Generic Functions
  - Function API
    - `AnsiFunction`
    - `Function`
    - `FunctionElement`
    - `GenericFunction`
    - `register_function()`
  - Selected "Known" Functions
    - `aggregate_strings`
    - `array_agg`
    - `char_length`
    - `coalesce`
    - `concat`
    - `count`
    - `cube`
    - `cume_dist`
    - `current_date`
    - `current_time`
    - `current_timestamp`
    - `current_user`
    - `dense_rank`
    - `grouping_sets`
    - `localtime`
    - `localtimestamp`
    - `max`
    - `min`
    - `mode`
    - `next_value`
    - `now`
    - `percent_rank`
    - `percentile_cont`
    - `percentile_disc`
    - `random`
    - `rank`
    - `rollup`
    - `session_user`
    - `sum`

Functionality of SQLAlchemy

Object-Relational Mapper (ORM)

SQLAlchemy's ORM allows you to map Python objects to database tables, providing a higher level of abstraction for database interactions. With the ORM, you can work with domain objects directly in your Python code, without needing to write raw SQL queries for common operations like CRUD (Create, Read, Update, Delete).

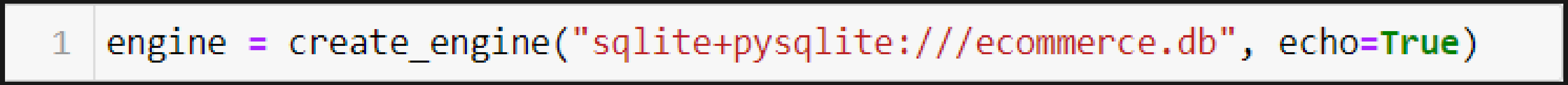


```python
class Product(Base):
    __tablename__ = 'products'

    id = Column(Integer, primary_key=True)
    name = Column(String(100), nullable=False)
    price = Column(Float, nullable=False)
    category = Column(String(50), nullable=False)

```

Functionality of SQLAlchemy

Database Connection Management

SQLAlchemy simplifies the process of managing database connections, including connection pooling and transaction management. It provides flexible configuration options for connecting to various database engines, such as SQLite, PostgreSQL, MySQL, and more.

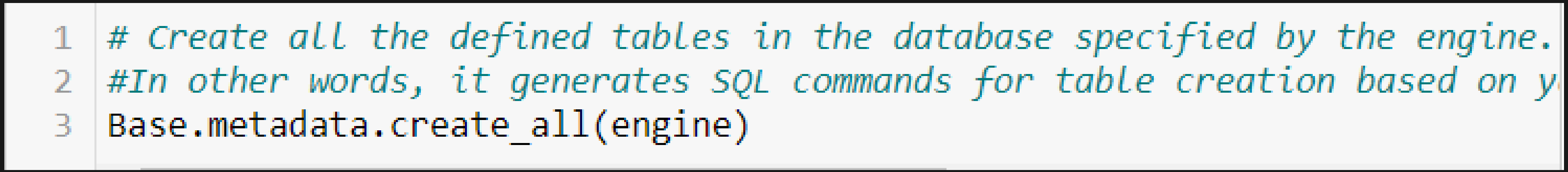


```python
engine = create_engine("sqlite+pysqlite:///ecommerce.db", echo=True)
```

Functionality of SQLAlchemy

Schema Definition and Migration

SQLAlchemy allows you to define database schemas using Python classes and constructs. You can define tables, columns, constraints, and relationships between tables using SQLAlchemy's declarative syntax. Additionally, SQLAlchemy provides tools for database schema migration, making it easier to manage changes to your database schema over time.

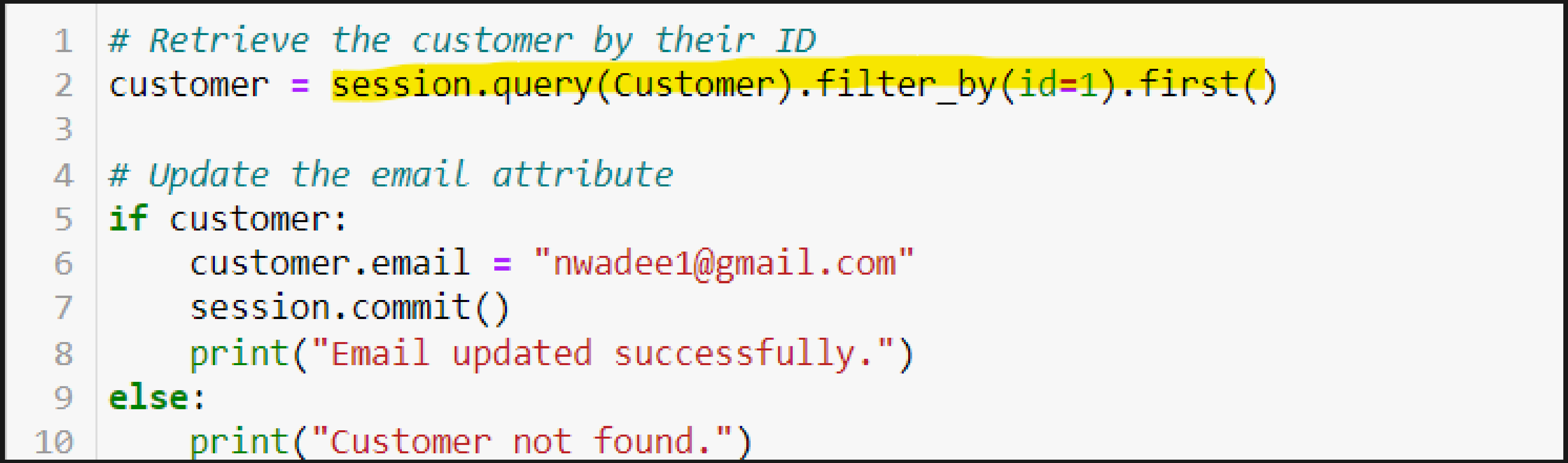


```python
# Create all the defined tables in the database specified by the engine.
#In other words, it generates SQL commands for table creation based on y
Base.metadata.create_all(engine)
```

Functionality of SQLAlchemy

Query Building and Execution

SQLAlchemy provides a comprehensive set of tools for building and executing database queries. Whether you're using the ORM or the Core API, SQLAlchemy allows you to construct complex queries with ease, including filtering, sorting, aggregating, and joining data from multiple tables.

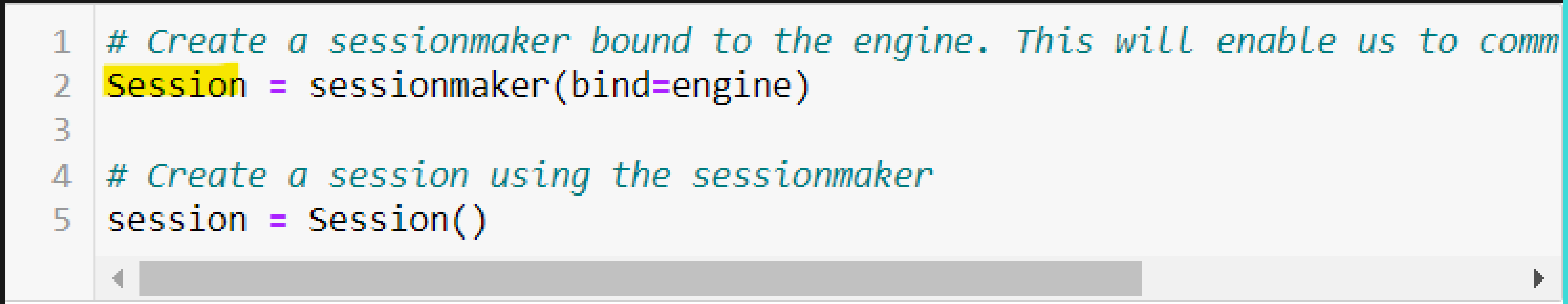


```python
# Retrieve the customer by their ID
customer = session.query(Customer).filter_by(id=1).first()

# Update the email attribute
if customer:
    customer.email = "nwadee1@gmail.com"
    session.commit()
    print("Email updated successfully.")
else:
    print("Customer not found.")
```

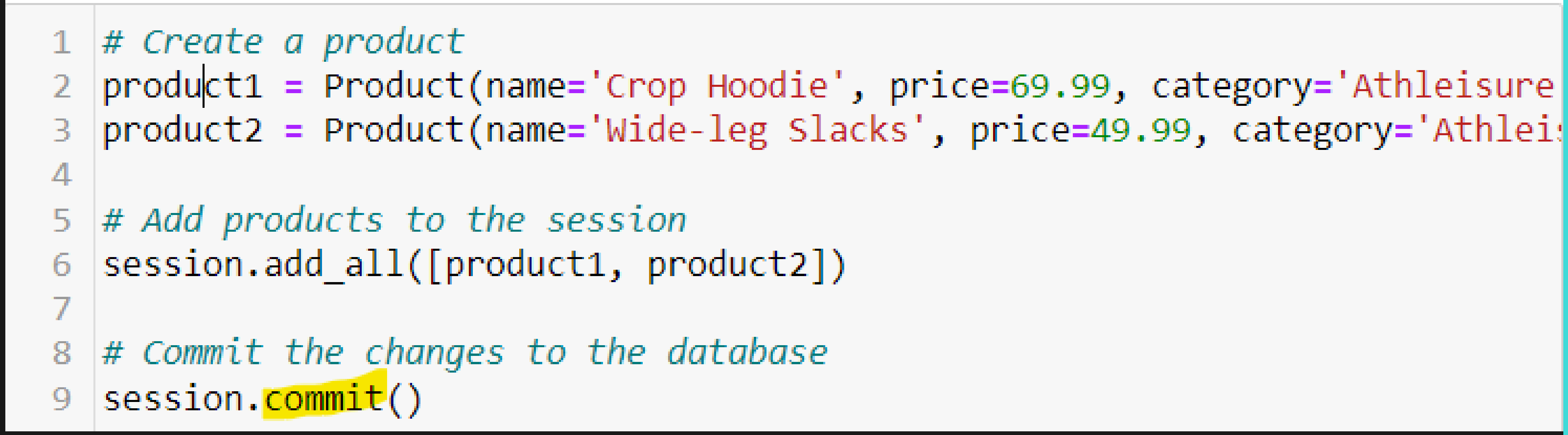
Functionality of SQLAlchemy

Transaction Management

SQLAlchemy supports transaction management, allowing you to group multiple database operations into a single atomic unit. This ensures that either all operations in the transaction are executed successfully, or none of them are, helping to maintain data integrity and consistency.



```python
# Create a sessionmaker bound to the engine. This will enable us to comm
Session = sessionmaker(bind=engine)

# Create a session using the sessionmaker
session = Session()
```

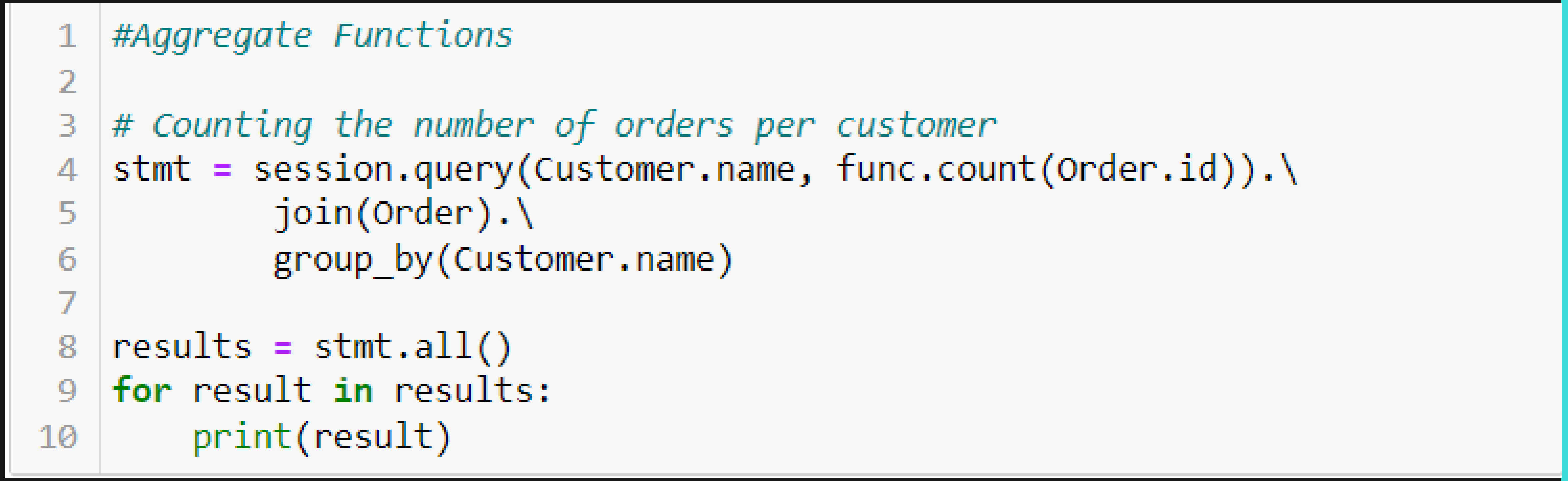


```python
# Create a product
product1 = Product(name='Crop Hoodie', price=69.99, category='Athleisure
product2 = Product(name='Wide-leg Slacks', price=49.99, category='Athlei

# Add products to the session
session.add_all([product1, product2])

# Commit the changes to the database
session.commit()
```

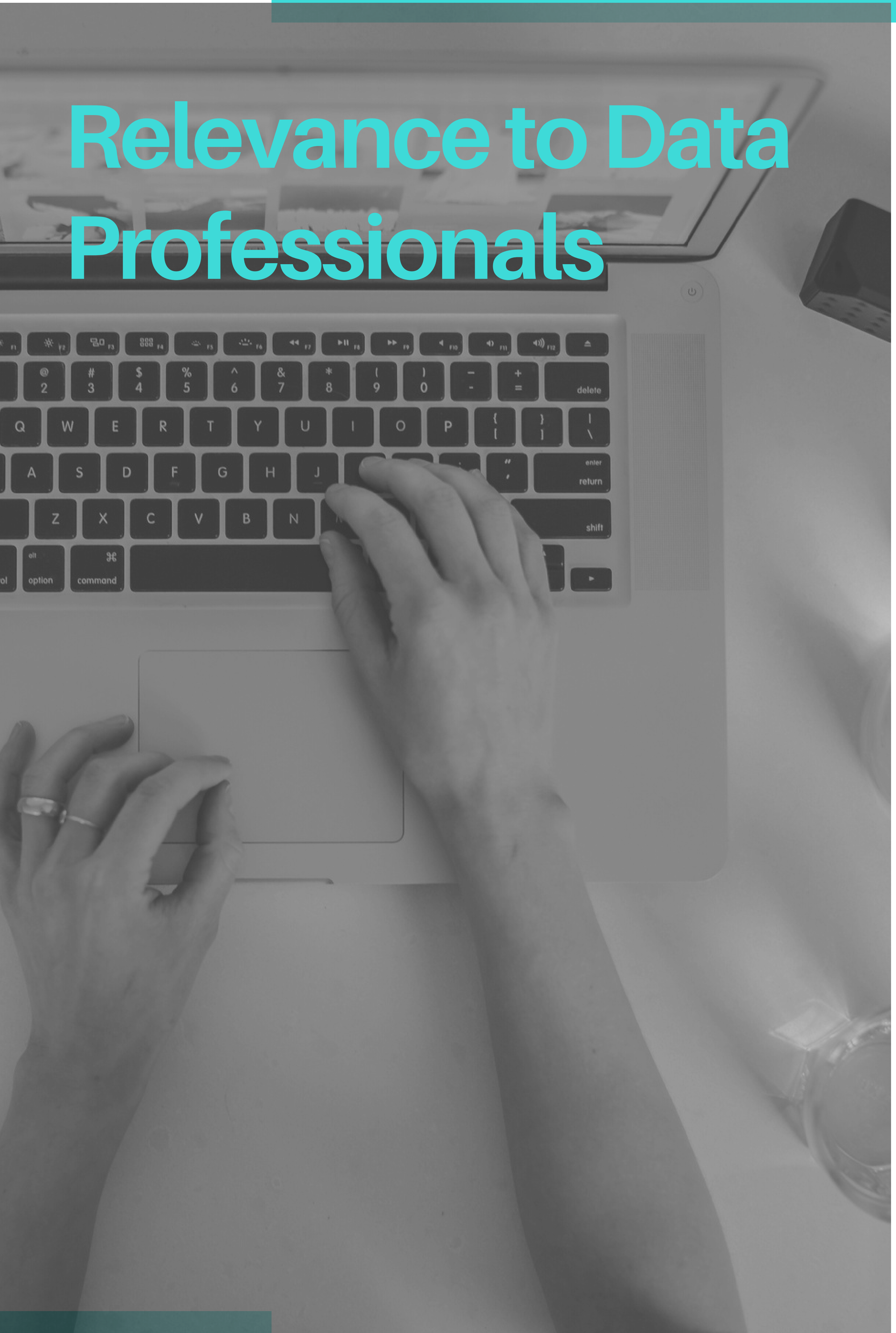
Functionality of SQLAlchemy

Advanced Query

SQLAlchemy includes advanced features such as support for advanced SQL constructs, including aggregations, joins, window functions, subqueries, and common table expressions (CTEs).



```python
#Aggregate Functions

# Counting the number of orders per customer
stmt = session.query(Customer.name, func.count(Order.id)).\
        join(Order).\
        group_by(Customer.name)

results = stmt.all()
for result in results:
    print(result)
```



Relevance to Data Professionals

1. Data Access and Manipulation
2. ORM for Object-Relational Mapping
3. Flexibility and Portability
4. Integration with Data Analysis Tools
5. Schema Management and Migration
6. Performance Optimization

Resources and References

SQLAlchemy Official Documentation
Link: https://docs.sqlalchemy.org/en/20/intro.html

Geeks for Geeks
Link: *https://www.geeksforgeeks.org/sqlalchemy-introduction/*

Tutorial