**Key Features of SQL Alchemy:**

SQLAlchemy consists of two distinct components, known as the **Core** and the **ORM**.

Some interesting features are things like eager-/lazy-loading, poly-morphic table layouts (in other words, having a table where the rows have a “type” column, then breaking the data elements into subclasses based on that column), caching, and database-platform independence.

* The Core is itself a fully featured SQL abstraction toolkit, providing a smooth layer of abstraction over a wide variety of DBAPI implementations and behaviors, as well as a SQL Expression Language which allows expression of the SQL language via generative Python expressions.
* The Object Relational Mapper is then an optional package which builds upon the Core. Many applications are built strictly on the Core, using the SQL expression system to provide succinct and exact control over database interactions.
* SqlAlchemy is high performing and accurate, well covered in tests, and deployed in thousands of environments. With virtually every major component in its second or third full iteration, SQLAlchemy 0.6 is roughly twice the speed of older 0.4 versions from just a few years ago, and versions 0.7 and 0.8 continue to improve.
* Built to conform to what DBAs demand, including the ability to swap out generated SQL with hand-optimized statements, full usage of bind parameters for all literal values, fully transactionalized and batched database writes using the Unit of Work pattern.
* SQLAlchemy places the highest value on not getting in the way of database and application architecture. Unlike many tools, it **never** "generates" schemas (not to be confused with issuing user-defined [DDL](http://en.wikipedia.org/wiki/Data_Definition_Language), in which it excels) or relies on naming conventions of any kind. SQLAlchemy supports the widest variety of database and architectural designs as is reasonably possible.
* SQLAlchemy includes dialects for SQLite, Postgresql, MySQL, Oracle, MS-SQL, Firebird, Sybase and others, most of which support multiple DBAPIs. Other dialects are published as external projects.
* The ORM standardizes on a "Declarative" configurational system that allows construction of user-defined classes inline with the table metadata they map to, in the same way most other object-relational tools provide. However this system is totally optional - at its core, the ORM considers the user-defined class, the associated table metadata, and the mapping of the two to be entirely separate. Through the use of the *mapper()* function, any arbitrary Python class can be mapped to a database table or view.
* n SQLAlchemy, primary and foreign keys are represented as sets of columns; truly composite behavior is implemented from the ground up. The ORM has industrial strength support for meaningful (non-surrogate) primary keys, including mutability and compatibility with ON UPDATE CASCADE, as well as explicit support for other common composite PK patterns such as "association" objects (many-to-many relationships with extra meaning attached to each association).
* Explicit support is available for single-table, concrete-table, and joined table inheritance. Polymorphic loading (that is, a query that returns objects of multiple descendant types) is supported for all three styles.

Source & Citation: https://www.sqlalchemy.org/features.html