

## Reference — Python DB-API

This is a reference for the material covered in the "Python DB-API" lesson.

---

### Python DB-API Quick Reference

For a full reference to the Python DB-API, see [the specification](#) and the documentation for specific database modules, such as [sqlite3](#) and [psycopg2](#).

#### **`module.connect(...)`**

Connect to a database. The arguments to **`connect`** differ from module to module; see the documentation for details. **`connect`** returns a **`Connection`** object or raises an exception.

For the methods below, note that you *don't* literally call (for instance) `Connection.cursor()` in your code. You make a `Connection` object, save it in a variable (maybe called `db`) and then call `db.cursor()`.

#### **`Connection.cursor()`**

Makes a **`Cursor`** object from a connection. Cursors are used to send SQL statements to the database and fetch results.

#### **`Connection.commit()`**

Commits changes made in the current connection. You must call **`commit`** before closing the connection if you want changes (such as inserts, updates, or deletes) to be saved. Uncommitted changes will be visible from your current connection, but not from others.

#### **`Connection.rollback()`**

Rolls back (undoes) changes made in the current connection. You must roll back if you get an exception if you want to continue using the same connection.

#### **`Connection.close()`**

Closes the connection. Connections are always implicitly closed when your program exits, but it's a good idea to close them manually especially if your code might run in a loop.

#### **`Cursor.execute(statement)`**

#### **`Cursor.execute(statement, tuple)`**

Execute an SQL statement on the database. If you want to substitute variables into the SQL statement, use the second form — see [the documentation](#) for details.

If your statement doesn't make sense (like if it asks for a column that isn't there), or if it asks the database to do something it can't do (like delete a row of a table that is referenced by other tables' rows) you will get an exception.

### **Cursor.fetchall()**

Fetch all the results from the current statement.

### **Cursor.fetchone()**

Fetch just one result. Returns a tuple, or **None** if there are no results.

---

## **psql Quick Reference**

The **psql** command-line tool is really powerful. There's a complete reference to it [in the PostgreSQL documentation](#).

To connect **psql** to a database running on the same machine (such as your VM), all you need to give it is the database name. For instance, the command **psql forum** will connect to the **forum** database.

From within **psql**, you can run any SQL statement using the tables in the connected database. Make sure to end SQL statements with a semicolon, which is not always required from Python.

You can also use a number of special **psql** commands to get information about the database and make configuration changes. The **\d posts** command shown in the video is one example — this displays the columns of the **posts** table.

Some other things you can do:

**\dt** — list all the tables in the database.

**\dt+** — list tables plus additional information (notably, how big each table is on disk).

**\H** — switch between printing tables in plain text vs. HTML.

---

## **Bleach documentation**

Read the documentation for Bleach here: <http://bleach.readthedocs.org/en/latest/>

---

## **Update and delete statements**

The syntax of the **update** and **delete** statements:

**update** *table* **set** *column = value* **where** *restriction* ;

**delete from** *table* **where** *restriction* ;

The **where** restriction in both statements works the same as in **select** and supports the same set of operators on column values. In both cases, if you leave off the **where** restriction, the update or deletion will apply to *all rows* in the table, which is usually not what you want.

---

## like operator

The **like** operator supports a simple form of text pattern-matching. Whatever is on the left side of the operator (usually the name of a text column) will be matched against the pattern on the right. The pattern is an SQL text string (so it's in '**single quotes**') and can use the **%** sign to match any substring, including the empty string.

If you are familiar with regular expressions, think of the **%** in **like** patterns as being like the regex **.\*** (dot star).

If you are more familiar with filename patterns in the Unix shell or Windows command prompt, **%** here is a lot like **\*(star)** in those systems.

For instance, for a table row where the column **fish** has the value '**salmon**', all of these restrictions would be true:

- **fish like 'salmon'**
- **fish like 'salmon%'**
- **fish like 'sal%'**
- **fish like '%n'**
- **fish like 's%n'**
- **fish like '%al%'**
- **fish like '%'**
- **fish like '%%%'**

And all of these would be false:

- **fish like 'carp'**
- **fish like 'salmonella'**
- **fish like '%b%'**

- fish like 'b%'
- fish like "

---

The term "spam" referring to junk posts comes from [Monty Python's "Spam" sketch](#). On the Internet, "spamming" was first used to mean [repetitious junk messages](#) intended to disrupt a group chat. Later, it came to refer to unsolicited ads on forums or email; and more recently to more-or-less any repetitious or uninvited junk message.