Getting started

This chapter is about how you connect to and work with databases in Python. It is a general purpose chapter. You will not learn the complete SQL language in one chapter. Instead, I will give you high level overview of some SQL commands and then we will look at how to connect to some of the most popular databases with Python. Most databases use the basic SQL commands in the same way, but they also may have some commands that are specific for that database backend or that just work slightly differently. Be sure to read your database documentation should you run into issues.

We will start the chapter by learning some very basic SQL syntax.

Basic SQL Syntax

SQL stands for Structured Query Language. It is basically the defacto language for interacting with databases and is a primitive computer programming language in its own right. In this section, we will learn how the basics of CRUD, which stands for Create, Read, Update and Delete. These are the most important functions you need to get started using a database. Of course, you will also need to know how to query, but we'll cover that as we go along as you will need to query in order to read, update or delete.

Creating a Table

The first thing you need in a database is a table. This is where you data will be stored and organized. Most of the time, you will have multiple tables where each table holds a sub-set of your data. Creating a table in SQL is easy. All you need to do is something like this:

```
CREATE TABLE table_name (
    id INTEGER,
    name VARCHAR,
```

```
    make VARCHAR
    model VARCHAR,

    year DATE,
    PRIMARY KEY (id)
    );
```

```
CREATE TABLE table_name (
    id INTEGER,
    name VARCHAR,
    make VARCHAR
    model VARCHAR,
    year DATE,
    PRIMARY KEY (id)
    );
```

This is pretty generic, but it works in most cases. The first item of note is that we have a bunch of capitalized words. These are the SQL commands. They don't usually need to be in caps, but we're doing that here to help you see them. I want to also note that each database supports slightly different commands. Most will have a CREATE TABLE, but the database column types might be different. You will note that in this example, we have INTEGER, VARCHAR and DATE data types. DATE can be called a lot of different things and VARCHAR can as well. Consult your documentation for what you will need to do.

Regardless, what we're doing in this example is creating a database with five columns. The first one is an **id** that we set as our primary key. It should not be NULL, but we don't specify that here as once again, each database backend does that differently or does it automatically for us. The other columns should be pretty self explanatory.

Inserting Data

Right now our database is empty. That's not very useful, so in this section we will learn how to add some data to it. Here's the general idea:

```
INSERT INTO table_name (id, name, make, model, year)
VALUES (1, 'Marly', 'Ford', 'Explorer', '2000');
```

SQL uses the INSERT INTO commands to add data to the specified database. You also specify why columns you are adding data too. When we create the

table, we can specify required column which could cause an error to be raised

if we didn't add data to a required column. However we didn't do that in our table definition earlier. It's just something to keep in mind. You will also receive an error if you pass the wrong data type, which I did for the year. I passed in a string or varchar instead of a date. Of course, each database requires a different date format, so you'll want to figure out what DATE even means for your database.

Updating Data

Let's say we made a typo in our previous INSERT. To fix that, we need to use SQL's UPDATE command:

```
UPDATE table_name
SET name='Chevrolet'
WHERE id=1;
```

The UPDATE command tells us which table we are going to update. Next we SET one or more columns to a new value. Finally we need to tell the database which row we want to update. We can use the WHERE command to tell the database we want to change the row that has an id of 1.

Reading Data

Reading data from our database is done via SQL's SELECT statement:

```
SELECT name, make, model
FROM table_name;
```

This will return all the rows from the database but the result will only contain three pieces of data: the name, make and model. If you want to grab all the data in the database, you can do this:

```
SELECT * FROM table_name;
```

The asterisk is a wildcard that tells SQL you want to grab all the columns. If you want to limit the scope of your select, you can add a WHERE to your query:

```
SELECT name, make, model
FROM table_name

WHERE year >= '2000-01-01' AND
      year <= '2006-01-01';
```

This will get back the name, make and model information for the years 2000 - 2006. There are a lot of other SQL commands to help you in your queries. Be sure to check out BETWEEN, LIKE, ORDER BY, DISTINCT and JOIN.

Deleting Data

Occasionally you will need to delete data from your database. Here's how:

```
DELETE FROM table_name
WHERE name='Ford';
```

This will delete all rows that have the name field set to 'Ford' from our table. If you wanted to delete the entire table, you would use the DROP statement:

```
DROP TABLE table_name;
```

Be careful when using DROP and DELETE as you can easily lose most or all of your data if you call the statement incorrectly. Always make sure you have a good, tested backup of your database.