

## CHAPTER 1

---

# Data tables in SQL

Most organizations keep their data as tables inside a relational database management system. Developers talk to those systems using a language called SQL (“Structured Query Language”).

Some relational database managers are pricey products you may have heard of before: Oracle, Microsoft SQL Server. Some are free: PostgreSQL or MySQL. These are server software that client programs talk to over the companies network.

There is a library, called `sqlite`, that lets us create files that hold tables. We can use SQL to create, edit, and browse those tables. `sqlite` is free, fast, and very easy to install. So we will use `sqlite` instead of a networked database management system.

If you look in your digital resources, you will find a file called `bikes.db`. I created this file using `sqlite`, and now you will use `sqlite` to access it.

In the terminal, get to the directory where `bikes.db` lives. To open the `sqlite` tool on that file:

```
> textbfsqlite3 bikes.db
```

(If your system complains that there is no `sqlite3` tool, you need to install `sqlite`. See this website: <https://sqlite.org/>)

Please follow along: type each command shown here into the terminal and see what happens.

We mostly run SQL commands in this tool, but there are a few non-SQL commands that all start with a period. To see the tables and their columns, you can run `.schema`:

```
sqlite> .schema
CREATE TABLE bike (bike_id int PRIMARY KEY, brand text, size int,
                    purchase_price real, purchase_date date, status text);
```

That is the SQL command that I used to create the `bike` table. You can see all the columns and their types.

You want to see all the rows of data in that table?

```
sqlite> select * from bike;
4997391|GT|57|269.61|2009-05-03|rented
5429447|Cannondale|50|215.91|2002-02-17|broken
5019171|Trek|58|251.17|1985-07-11|rented
3000288|Cannondale|57|211.08|1993-01-05|broken
880965|GT|52|281.75|1995-08-02|available
...
```

You will see 1000 rows of data!

The SQL language is not case-sensitive, so you can also write it like this:

```
sqlite> SELECT * FROM BIKE;
```

Often you will see SQL with just the SQL keywords in all caps:

```
sqlite> SELECT * FROM bike;
```

The semicolon is not part of SQL, but it tells sqlite that you are done writing a command and that it should be executed.

SQL lets you choose which columns you would like to see:

```
sqlite> SELECT bike_id, brand FROM bike;
4997391|GT
5429447|Cannondale
5019171|Trek
3000288|Cannondale
...
```

Using WHERE, SQL lets you choose which rows you would like to see:

```
sqlite> SELECT * FROM bike WHERE purchase_date > '2009-01-01' AND brand = 'GT';
4997391|GT|57|269.61|2009-05-03|rented
326774|GT|56|165.0|2009-06-27|available
264933|GT|52|302.43|2009-07-09|available
5931243|GT|55|173.56|2009-11-26|rented
4819848|GT|51|221.71|2009-12-11|rented
9347713|GT|52|232.32|2009-06-13|available
3019205|GT|58|262.94|2009-08-22|available
```

Using DISTINCT, SQL lets you get just one copy of each value:

```
sqlite> SELECT DISTINCT status FROM bike;
rented
broken
available
```

```
Busted
Flat tire
good
out
Rented
```

You can also edit these rows. For example, if you wanted every status that is Busted to be changed to broken. You can use an UPDATE statement:

```
sqlite> UPDATE bike SET status='broken' WHERE status='Busted';
sqlite> SELECT DISTINCT status FROM bike;
rented
broken
available
```

```
Flat tire
good
out
Rented
```

You can insert new rows:

```
sqlite> INSERT INTO bike (bike_id, brand, size, purchase_price, purchase_date, status)
...> VALUES (1, 'GT', 53, 123.45, '2020-11-13', 'available');
sqlite> SELECT * FROM bike WHERE bike_id = 1;
1|GT|53|123.45|2020-11-13|available
```

You can delete rows:

```
sqlite> DELETE FROM bike WHERE bike_id = 1;
sqlite> SELECT * FROM bike WHERE bike_id = 1;
```

To get out of sqlite, type `.exit`.

## Exercise 1 SQL Query

Execute an SQL query that returns the `bike_id` (no other columns) of every Trek bike that cost more than \$300.

*Working Space*

*Answer on Page 7*

## 1.1 Using SQL from Python

The people behind `sqlite` created a library for Python that lets you execute SQL and fetch the results from inside a python program.

Let's create a simple program that fetches and displays the bike ID and purchase date of every Trek bike that cost more than \$300.

Create a file called `report.py`:

```
import sqlite3 as db

con = db.connect('bikes.db')
cur = con.cursor()

cur.execute("SELECT bike_id, purchase_date FROM bike WHERE purchase_price > 330 AND brand='Trek'")
rows = cur.fetchall()

today = datetime.date.today()
for row in rows:
    print(f"Bike {row[0]}, purchased {row[1]}")

con.close()
```

When you execute it, you should see:

```
> python3 report.py
Bike 4128046, purchased 2007-08-06
Bike 7117808, purchased 1995-03-12
Bike 7176903, purchased 1986-07-03
Bike 827899, purchased 2009-03-14
```

Bike 363983, purchased 1970-08-16

---

*This is a draft chapter from the Kontinua Project. Please see our website (<https://kontinua.org/>) for more details.*



# Answers to Exercises

## Answer to Exercise 1 (on page 4)

```
SELECT bike_id FROM bike WHERE purchase_price > 330 AND brand='Trek'
```

