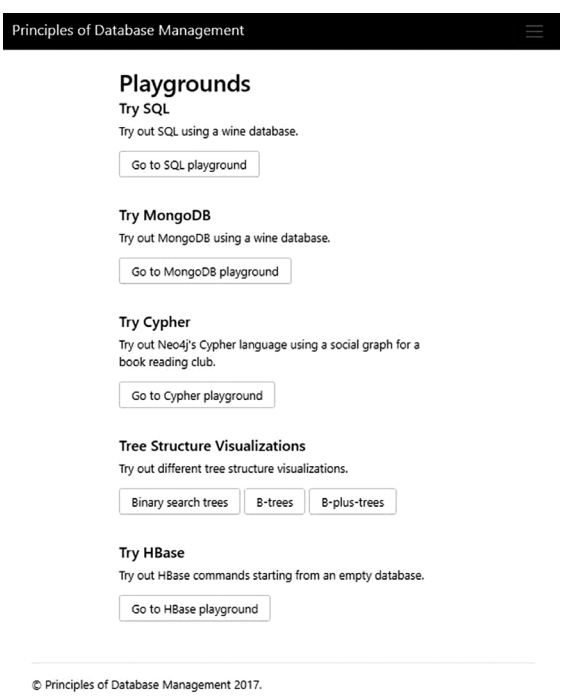
APPENDIX USING THE ONLINE ENVIRONMENT

Here, we will show you how you can use the online environment for this book to play around with several types of database management systems and database query languages.

How to Access the Online Environment

1. Navigate to www.pdbmbook.com/playground.
2. If this is your first time accessing the environment, you'll need to register first. Click on the "register" link and enter your details.
3. You'll receive a confirmation email with a unique link you need to open to verify your account.
4. Afterwards, you can log-in and see the following overview of interactive environments:[1]



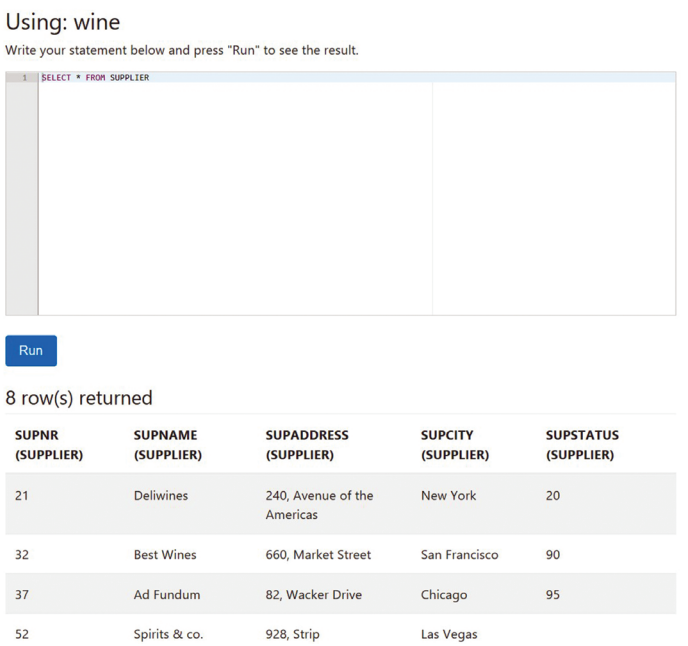
[1] The online environment may be further expanded in the future, so the actual overview could be somewhat different from the one depicted here.

Environment: Relational Databases and SQL

In Chapter 7 on the Structured Query Language, a wine database was introduced. You can follow along with the queries by navigating the SQL environment.

For instance, to select all information from the SUPPLIER table, you can execute:

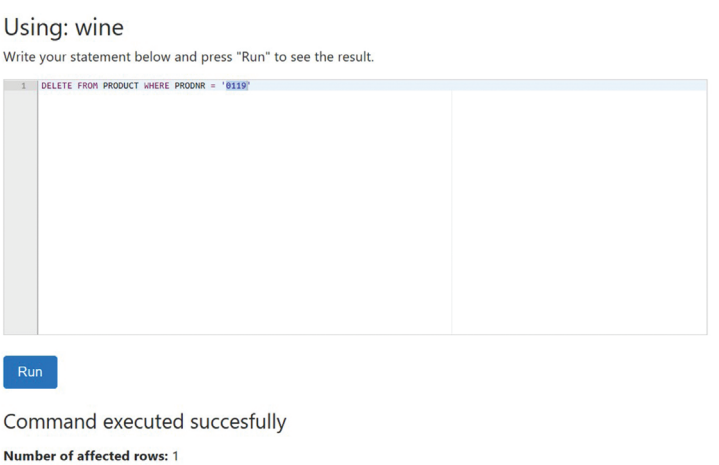
```
SELECT * FROM SUPPLIER
```



SUPNR (SUPPLIER)	SUPNAME (SUPPLIER)	SUPADDRESS (SUPPLIER)	SUPCITY (SUPPLIER)	SUPSTATUS (SUPPLIER)
21	Deliwines	240, Avenue of the Americas	New York	20
32	Best Wines	660, Market Street	San Francisco	90
37	Ad Fundum	82, Wacker Drive	Chicago	95
52	Spirits & co.	928, Strip	Las Vegas	

Feel free to execute INSERT, UPDATE, and DELETE queries as well. The following example removes the product tuple with product number 0119 from the PRODUCT table:

```
DELETE FROM PRODUCT WHERE PRODNR = '0119'
```



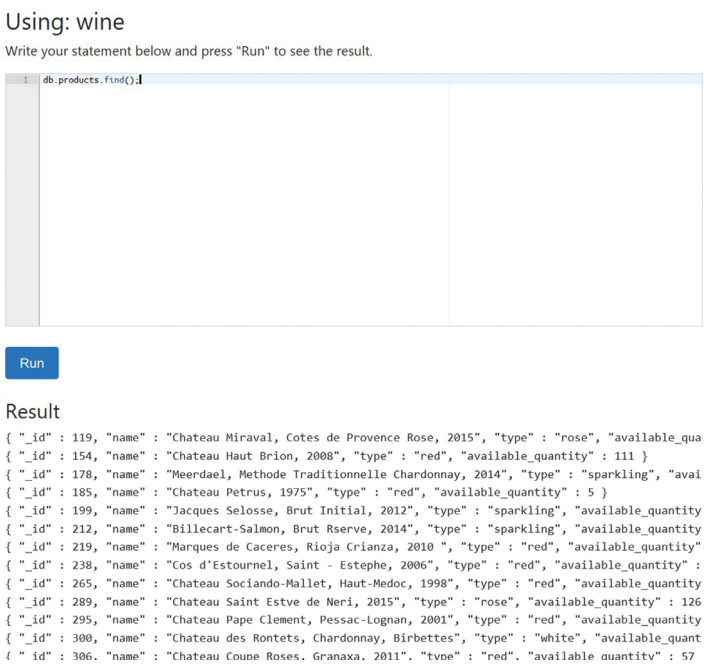
If you want to start over with a fresh database (helpful if you just deleted a complete table), you can press the “reset” link to reset the database to its initial state.

Environment: MongoDB

In Chapter 11 on NoSQL databases, we discussed MongoDB as an example of a document store. The respective online environment contains a MongoDB version of the wine database, which you can query through its JavaScript shell. Some interesting commands to play around with are the following.

Retrieve all documents in the "products" collection:

```
db.products.find();
```



Retrieve all documents in the "products" collection where the id matches 119:

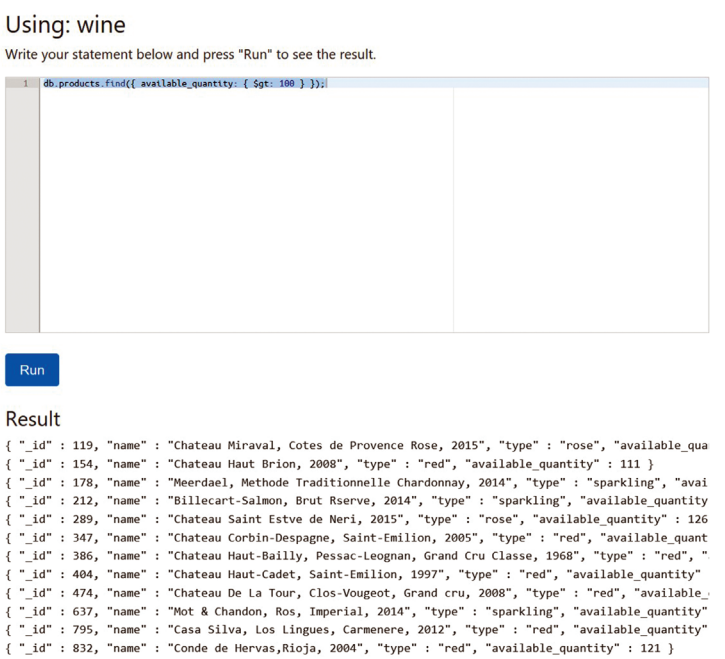
```
db.products.find({ _id: 119 });
```

Retrieve all documents in the “products” collection where the type matches “rose”:

```
db.products.find({  type: 'rose' });
```

Retrieve all documents in the “products” collection where the available quantity is greater than 100:

```
db.products.find({  available_quantity: {  $gt: 100 } });
```



Aggregate some documents in the "products" collection as follows: take all products with an available quantity greater than 100, group these by their "type", and create a "total" field by summing "1" per record in each group. Finally, sort descending on "total":

```
db.products.aggregate([
    { $match: { available_quantity: { $gt: 100} } },
    { $group: { _id: "$type", total: { $sum: 1 } } },
    { $sort: { total: -1 } }
]);
```

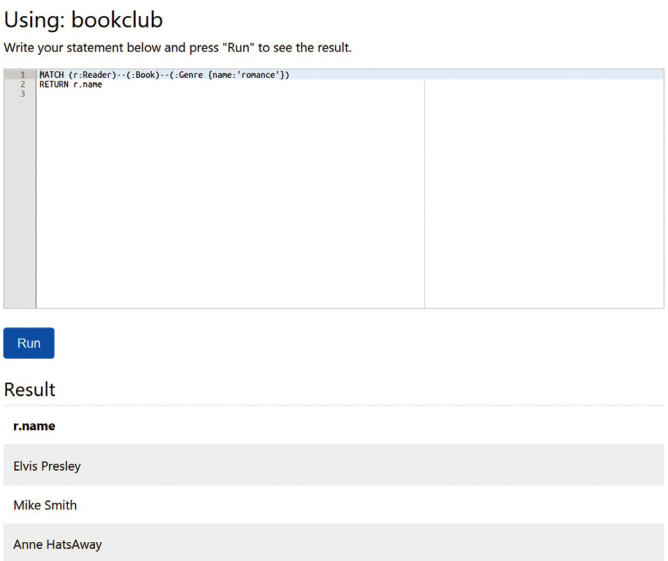
If you want to start over with a fresh MongoDB instance, you can press the "reset" link to reset the database to its initial state.

Environment: Neo4j and Cypher

In Chapter 11 on NoSQL databases, we discussed Neo4j as an example of a graph database. The online environment contains a Neo4j database for our book-reading club. You can use the same queries as the ones in the chapter to follow along.

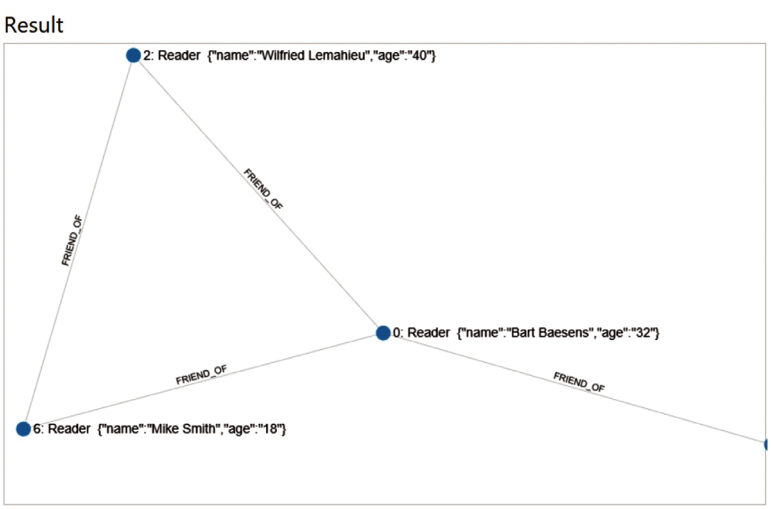
For instance: who likes romance books?

```
MATCH (r:Reader)--(:Book)--(:Genre {name:'romance'})
RETURN r.name
```



Show Bart and Bart's friends that liked humor books:

```
MATCH (me:Reader)--(friend:Reader)--(b:Book)--(g:Genre)
WHERE g.name = 'humor'
  AND me.name = 'Bart Baesens'
RETURN me, friend
```



If you want to start over with a fresh Neo4j instance, you can press the "reset" link to reset the database to its initial state.

Environment: Tree Structure Visualizations

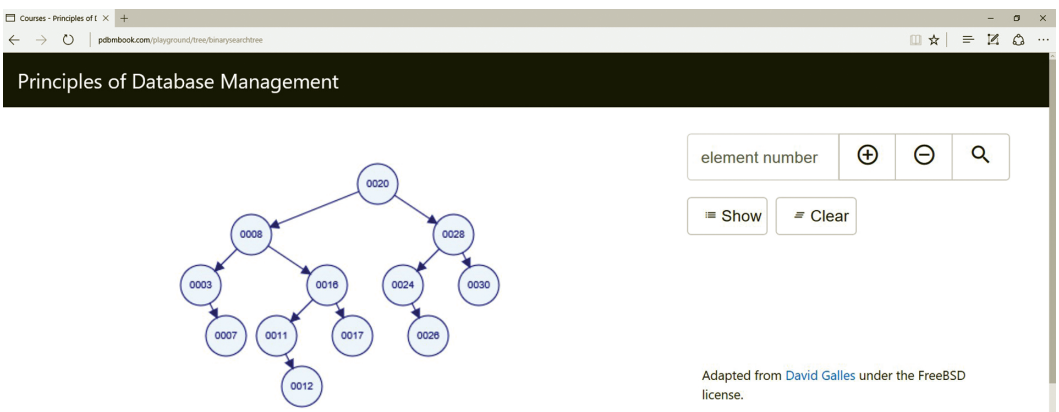
This environment provides visualization of several tree-based data structures as discussed in Chapter 12 (i.e., binary search trees, B-trees, and B^+-trees).

A binary search tree is a physical tree structure in which each node has at most two children. Each tree node contains a search key value and (at most) two pointers to children. Both children are the root nodes of subtrees, with one subtree only containing key values that are lower than the key value in the original node, and the other subtree only containing key values that are higher.

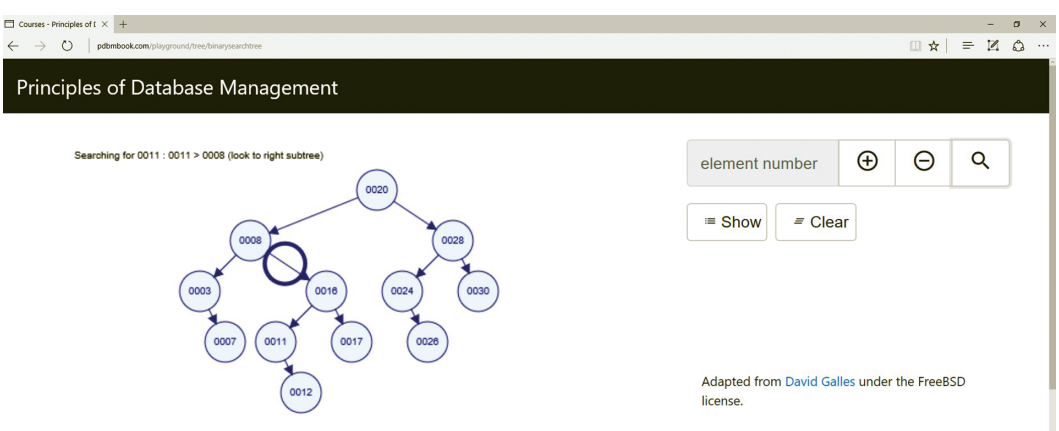
To see how this works, you can try inserting the following elements individually in the online environment:

20, 8, 28, 3, 16, 24, 30, 7, 11, 17, 26, 12

You will obtain a similar view to the tree shown in the chapter:



Navigating a binary search tree is very efficient, as half of the search key values can be skipped with every step, rather than linearly navigating all key values. To illustrate this, try using the "find" button to look for key 11. An animation will play showing how the tree is traversed:



The performance could be increased even further if each node would contain more than one key value and more than two children. In that case, with an equal total number of key values, the height of the tree would be reduced and therefore the average and maximal number of steps would be lower.

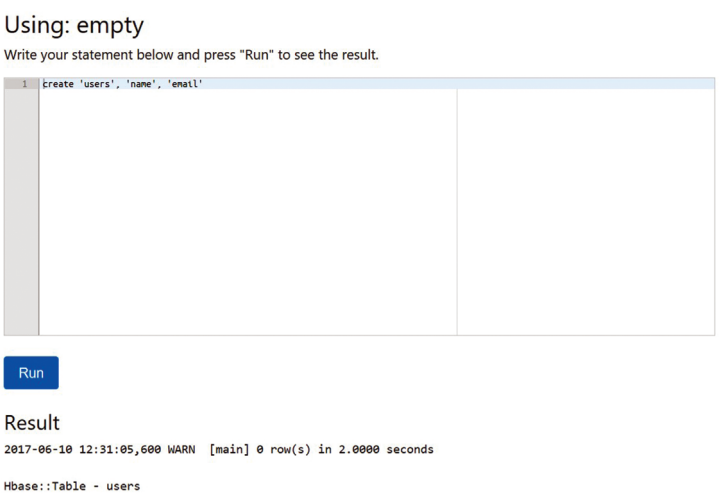
This exact consideration is at the basis of the B-tree concept. B-trees and B^+-trees can be considered as variations of search trees that are explicitly designed for hard disk storage. Every node contains a set of search key values, a set of tree pointers that refer to child nodes, and a set of data pointers that refer to data records that correspond to the search key values. The data records are stored separately and are not part of the B-tree. You can play around with B-trees and B^+-trees in the environment as well.

Environment: HBase

In Chapter 19 on Big Data, we discussed HBase as a first, key–value based data store that runs on top of Hadoop. In the online environment, you can experiment with the HBase shell and follow along with the queries in the chapter.

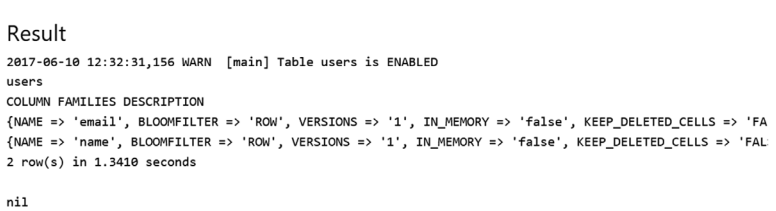
We start with an empty HBase set-up in this environment, so let's create a simple HBase table to store and query users using the HBase shell. Let's start by creating our "users" table with two column families (note: the HBase environment can take some time to parse each command!):

```
create 'users', 'name', 'email'
```



Describe the table:

```
describe 'users'
```



List this table (the table itself, not its contents):

```
list 'users'
```

We can now insert values. Since HBase represents data as a multidimensional map, we store values using "put" one by one by specifying the row key, column family:qualifier, and the value itself. Note you can enter multiple statements by just entering them line-by-line:

```
put 'users', 'seppe', 'name:first', 'Seppe'
put 'users', 'seppe', 'name:last', 'vanden Broucke'
put 'users', 'seppe', 'email', 'seppe.vandenbroucke@kuleuven'
```

Now list the full contents of this table using the "scan" command:

```
scan 'users'
```

Result

```
2017-06-10 12:36:29,150 WARN  [main] ROW  COLUMN+CELL
 seppe column=email:, timestamp=1497112543342, value=seppe.vandenbroucke@kuleuven
 seppe column=name:first, timestamp=1497112543161, value=Seppe
 seppe column=name:last, timestamp=1497112543274, value=vanden Broucke
1 row(s) in 0.6330 seconds

nil
```

Give the information for row key "seppe" only:

```
get 'users', 'seppe'
```

Result

```
2017-06-10 12:38:28,163 WARN  [main] COLUMN  CELL
 email: timestamp=1497112543342, value=seppe.vandenbroucke@kuleuven
 name:first timestamp=1497112543161, value=Seppe
 name:last timestamp=1497112543274, value=vanden Broucke
3 row(s) in 0.7000 seconds

nil
```

We can change the email value by just running put again:

```
put 'users', 'seppe', 'email', 'seppe@kuleuven.be'
```

Let's now retrieve this row again, but only for the column family "email":

```
get 'users', 'seppe', 'email'
```

Result

```
2017-06-10 12:39:41,915 WARN  [main] 0 row(s) in 0.8180 seconds

nil
COLUMN  CELL
 email: timestamp=1497112786182, value=seppe@kuleuven.be
1 row(s) in 0.0870 seconds

nil
```

We can delete all values pertaining to row key “seppe” in “users” as follows and scan the table to confirm that the data are deleted:

```
deleteall 'users', 'seppe'
scan 'users'
```

Result

```
2017-06-10 12:41:02,753 WARN  [main] 0 row(s) in 0.8420 seconds

nil
ROW  COLUMN+CELL
0 row(s) in 0.0640 seconds

nil
```

If you want to start over with a fresh HBase instance, you can press the “reset” link to reset the database to its initial, empty state.