MORE ITERATORS, GENERATORS AND SQL 11

DATA C88C

November 14, 2023

1 More Iterators and Generators!

1.1 Questions

1. To make the `Link` class iterable, implement the `LinkIterator` class.

```
class Link:
    empty = ()
    def __init__(self, first, rest=empty):
        self.first = first
        self.rest = rest
    def __iter__(self):
        return LinkIterator(self)
```

```
class LinkIterator:
    def __init__(self, link):
```

Solution:

```
        self.link = link
```

```
    def __iter__(self):
```

Solution:

```
        return self
```

```
    def __next__(self):
```

Solution:

```
        if self.link is Link.empty:
            raise StopIteration
        val = self.link.first
        self.link = self.link.rest
        return val
```

2. Implement `accumulate`, which takes in an `iterable` and a function `f` and yields each accumulated value from applying `f` to the running total and the next element.

```
from operator import add, mul

def accumulate(iterable, f=add):
    """Return running totals

    >>> list(accumulate([1,2,3,4,5]))
    [1, 3, 6, 10, 15]
    >>> list(accumulate([1,2,3,4,5], mul))
    [1, 2, 6, 24, 120]
    """
    it = iter(iterable)
```

Solution:

```
    total = next(it)
    yield total
    for element in it:
        total = f(total, element)
        yield total
```

2 SQL Introduction

In Python, we wrote programs using **imperative programming** – where code is written as a set of instructions for the computer. In contrast, with **declarative programming** ourcode declares *what* result we want, not *how* to compute it.

SQL is an example of a declarative programming language. Statements do not describe computations directly, but instead describe the desired result of some computation. It is the role of the query interpreter of the database system to plan and perform a computational process to produce such a result.

In SQL, data is organized into *tables*. A table has a fixed number of named **columns**. A **row** of the table represents a single data record and has one **value** for each column. For example, we have a table named `records` that stores information about the employees at a small company[1]. Each of the eight rows represents an employee.

name	division	title	salary	supervisor
Ben Bitdiddle	Computer	Wizard	60000	Oliver Warbucks
Alyssa P Hacker	Computer	Programmer	40000	Ben Bitdiddle
Cy D Fect	Computer	Programmer	35000	Ben Bitdiddle
Lem E Tweakit	Computer	Technician	25000	Ben Bitdiddle
Louis Reasoner	Computer	Programmer Trainee	30000	Alyssa P Hacker
Oliver Warbucks	Administration	Big Wheel	150000	Oliver Warbucks
Eben Scrooge	Accounting	Chief Accountant	75000	Oliver Warbucks
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

Solution: Video walkthrough

3 Creating Tables

We can use a `SELECT` statement to create tables. The following statement creates a table with a single row, with columns named "first" and "last":

```
sqlite> SELECT "Ben" AS first, "Bitdiddle" AS last;
Ben|Bitdiddle
```

Given two tables with the same number of columns, we can combine their rows into a larger table with `UNION`:

```
sqlite> SELECT "Ben" AS first, "Bitdiddle" AS last UNION
   ...> SELECT "Louis",        "Reasoner";
Ben|Bitdiddle
Louis|Reasoner
```

[1] Example adapted from Structure and Interpretation of Computer Programs

To save a table, use `CREATE TABLE` and a name. Here we're going to create the table of employees from the previous section and assign it to the name `records`:

```
sqlite> CREATE TABLE records AS
   ...>   SELECT "Ben Bitdiddle" AS name, "Computer" AS division,
   ...>     "Wizard" AS title, 60000 AS salary,
   ...>     "Oliver Warbucks" AS supervisor UNION
   ...>   SELECT "Alyssa P Hacker", "Computer",
   ...>     "Programmer", 40000, "Ben Bitdiddle" UNION ... ;
```

We can `SELECT` specific values from an existing table using a `FROM` clause. This query creates a table with two columns, with a row for each row in the `records` table:

```
sqlite> SELECT name, division FROM records;
Alyssa P Hacker|Computer
Ben Bitdiddle|Computer
Cy D Fect|Computer
Eben Scrooge|Accounting
Lem E Tweakit|Computer
Louis Reasoner|Computer
Oliver Warbucks|Administration
Robert Cratchet|Accounting
```

The special syntax `SELECT *` will select all columns from a table. It's an easy way to print the contents of a table.

```
sqlite> SELECT * FROM records;
Alyssa P Hacker|Computer|Programmer|40000|Ben Bitdiddle
Ben Bitdiddle|Computer|Wizard|60000|Oliver Warbucks
Cy D Fect|Computer|Programmer|35000|Ben Bitdiddle
Eben Scrooge|Accounting|Chief Accountant|75000|Oliver Warbucks
Lem E Tweakit|Computer|Technician|25000|Ben Bitdiddle
Louis Reasoner|Computer|Programmer Trainee|30000|Alyssa P Hacker
Oliver Warbucks|Administration|Big Wheel|150000|Oliver Warbucks
Robert Cratchet|Accounting|Scrivener|18000|Eben Scrooge
```

We can choose which columns to show in the first part of the `SELECT`, we can filter out rows using a `WHERE` clause, and sort the resulting rows with an `ORDER BY` clause. In general the syntax is:

```
SELECT [columns] FROM [tables]
  WHERE [condition] ORDER BY [criteria];
```

For instance, the following statement lists all information about employees with the "Programmer" title.

```
sqlite> SELECT * FROM records WHERE title = "Programmer";
Alyssa P Hacker|Computer|Programmer|40000|Ben Bitdiddle
Cy D Fect|Computer|Programmer|35000|Ben Bitdiddle
```

The following statement lists the names and salaries of each employee under the accounting division, sorted in **descending** order by their salaries.

```
sqlite> SELECT name, salary FROM records
   ...>   WHERE division = "Accounting" ORDER BY -salary;
Eben Scrooge|75000
Robert Cratchet|18000
```

Equivalently, we can use the `ASC` and `DESC` keywords to sort the values in ascending and descending order, respectively. By default, `ORDER BY` sorts values in ascending order. To achieve the same result as above, we would write:

```
sqlite> SELECT name, salary FROM records
   ...>   WHERE division = "Accounting" ORDER BY salary DESC;
Eben Scrooge|75000
Robert Cratchet|18000
```

Note that all valid SQL statements must be terminated by a semicolon (`;`). Additionally, you can split up your statement over many lines and add as much whitespace as you want, as long as you have the semicolon at the end. But keep in mind that having consistent indentation and line breaking does make your code a lot more readable to others (and your future self)!

3.1 Questions

1. Write a query that outputs the names of employees that Oliver Warbucks directly supervises.

Solution:
```
select name from records where supervisor = "Oliver
   Warbucks";
```

2. Write a query that outputs all information about self-supervising employees.

Solution:
```
select * from records where name = supervisor;
```

3. Write a query that outputs the names of all employees with salary greater than 50000 in alphabetical order.

Solution:
```
select name from records where salary > 50000 order by name
   ;
```

4 Joins

Suppose we have another table `meetings` which records the divisional meetings.

`meetings`

division	day	time
Accounting	Monday	9am
Computer	Wednesday	4pm
Administration	Monday	11am
Administration	Wednesday	4pm

Data are combined by joining multiple tables together into one, a fundamental operation in database systems. There are many methods of joining, all closely related, but we will focus on just one method (the inner join) in this class.

When tables are joined, the resulting table contains a new row for each combination of rows in the input tables. If two tables are joined and the left table has m rows and the right table has n rows, then the joined table will have mn rows. Joins are expressed in SQL by separating table names by commas in the `FROM` clause of a `SELECT` statement.

```
sqlite> SELECT name, day FROM records, meetings;
Ben Bitdiddle | Monday
```

```
Ben Bitdiddle | Wednesday
...
Alyssa P Hacker | Monday
...
```

Tables may have overlapping column names, and so we need a method for disambiguating column names by table. A table may also be joined with itself, and so we need a method for disambiguating tables. To do so, SQL allows us to give aliases to tables within a `FROM` clause using the keyword `AS` and to refer to a column within a particular table using a dot expression. In the example below we find the name and title of Louis Reasoner's supervisor.

```
sqlite> SELECT b.name, b.title FROM records AS a, records AS b
   ...>   WHERE a.name = "Louis Reasoner" AND
   ...>         a.supervisor = b.name;
Alyssa P Hacker | Programmer
```

Solution: Video walkthrough

4.1 Questions

1. Write a query that creates a table with columns: employee name, employee salary, supervisor name, and supervisor's salary, containing all supervisors who earn more than twice as much as the employee.

   Solution:
   ```
   select e.name, e.salary, s.name, s.salary
     from records as e, records as s
     where e.supervisor = s.name and e.salary * 2 < s.salary;
   ```

2. Write a query that outputs the names of employees whose supervisor is in a different division.

   Solution:
   ```
   select e.name from records as e, records as s
     where e.supervisor = s.name and e.division != s.division;
   ```

3. Write a query that outputs the meeting days and times of all employees directly supervised by Oliver Warbucks.

Solution:

```
select m.day, m.time from records as r, meetings as m
  where r.division = m.division and
        r.supervisor = "Oliver Warbucks";
```

4.2 Extra Questions

1. A middle manager is a person who is both supervising someone and is supervised by someone different. Write a query that outputs the names of all middle managers.

Solution:

```
select b.name from records as a, records as b
  where a.supervisor = b.name and b.supervisor != b.name;
```

2. What is the output of the query in the previous part? Explain the output you get.

Solution:

```
Alyssa P Hacker
Ben Bitdiddle
Ben Bitdiddle
Ben Bitdiddle
Eben Scrooge
```

There are multiple people with Ben Bitdiddle as supervisor, and joining tables together does not remove these duplicates.

3. Write a query that results in the names of all employees that have a meeting on the same day as their supervisor.

Solution:

```
select e.name from records as e, records as s, meetings as
   em, meetings as sm
  where e.supervisor = s.name and em.day = sm.day and
        e.division = em.division and s.division = sm.
           division;
```

5 Aggregation

So far, we have joined and manipulated individual rows using `SELECT` statements. But we can also perform aggregation operations over multiple rows with the same `SELECT` statements.

We can use the `MAX`, `MIN`, `COUNT`, and `SUM` functions to retrieve more information from our initial tables.

If we wanted to find the name and salary of the employee who makes the most money, we might say

```
sqlite> SELECT name, MAX(salary) FROM records;
Oliver Warbucks|150000
```

Using the special `COUNT(*)` syntax, we can count the number of rows in our table to see the number of employees at the company.

```
sqlite> SELECT COUNT(*) from RECORDS;
9
```

These commands can be performed on specific sets of rows in our table by using the `GROUP BY [column name]` clause. This clause takes all of the rows that have the same value in `column name` and groups them together.

We can find the miniumum salary earned in each division of the company.

```
sqlite> SELECT division, MIN(salary) FROM records GROUP BY
   division;
Computer|25000
Administration|25000
Accounting|18000
```

These groupings can be additionally filtered by the `HAVING` clause. In contrast to the `WHERE` clause, which filters out individual rows, the `HAVING` clause filters out entire groups. `HAVING` may only be used in conjunction with a `GROUP BY` clause.

To find all titles that are held by more than one person, we say

```
sqlite> SELECT title FROM records GROUP BY title HAVING count(*)
   > 1;
Programmer
```

5.1 Questions

1. Write a query that outputs each supervisor and the sum of salaries of all of each supervisor's employees.

Solution:

```
select supervisor, sum(salary) from records group by
   supervivor;
```

2. Write a query that outputs all salaries that appear more than once in the employee records.

Solution:

```
select salary from records group by salary having count(*)
   > 1;
```