

## COMPUTER SCIENCE 61A

April 21, 2016

---

**1 Introduction**

---

In Python, we wrote programs using **imperative programming** – a programming style where code is written as a set of instructions for the computer. In contrast, with **declarative programming** our code 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<sup>1</sup>. The table has five columns, and each of the nine rows represents an employee.

<b>Name</b>	<b>Division</b>	<b>Title</b>	<b>Salary</b>	<b>Supervisor</b>
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
DeWitt Aull	Administration	Secretary	25000	Oliver Warbucks
Eben Scrooge	Accounting	Chief Accountant	75000	Oliver Warbucks
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

---

<sup>1</sup>Example adapted from Structure and Interpretation of Computer Programs

---

## 2 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
```

To save a table for use later, use `create table` and the name we want to give the table. 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 rows 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  
DeWitt Aull|Administration  
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 any 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
DeWitt Aull|Administration|Secretary|25000|Oliver Warbucks
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, 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
```

## 2.1 Questions

---

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



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

### 3.1 Questions

---

1. Write a query that creates a table with columns: `employee`, `salary`, `supervisor` and `supervisor's salary`, containing all supervisors who earn more than twice as much as the employee.
2. Write a query that outputs the names of employees whose supervisor is in a different division.
3. Write a query that outputs the meeting days and times of all employees directly supervised by Oliver Warbucks.

### 3.2 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.





## 5 Recursive Queries

### 5.1 With Statements

The `select` statement can optionally include a `with` clause that generates and names *local* tables used in computing the final result. These local tables cannot be used outside of the `select` statement, and they can be thought of as “helper” tables. The full syntax of a `select` statement has the following form:

```
with [local-tables] select [columns] from [tables]
  where [condition] order by [criteria]
```

For example, you can use a `with` statement to create and immediately use a new table to compute the final result.

```
sqlite> with
...>   schedule(day, dresscode) as (
...>     select "Monday",    "Sports"    union
...>     select "Tuesday",   "Drag"     union
...>     select "Wednesday", "Regular"   union
...>     select "Thursday",  "Throwback" union
...>     select "Friday",    "Casual"
...>   )
...> select a.name, b.dresscode from
...>   records as a, schedule as b, meetings as c
...>   where a.division = c.division and
...>   b.day = c.day order by a.name;
```

```
Alyssa P Hacker|Regular
Ben Bitddiddle|Regular
Cy D Fect|Regular
DeWitt Aull|Sports
DeWitt Aull|Throwback
Eben Scrooge|Sports
Lem E Tweakit|Regular
Louis Reasoner|Regular
Oliver Warbucks|Sports
Oliver Warbucks|Throwback
Robert Cratchet|Sports
```

If you try selecting from `schedule` outside, like

```
sqlite> select * from schedule;
Error: no such table: schedule
```

you’ll find that you cannot reference it because `schedule` was not made globally.

---

## 5.2 Recursive Selects

---

We can create recursive table definitions using the `with` syntax.

Let's create a table of natural numbers from 0 to 5 (inclusive). We want to employ the same thought process as we did with the recursive functions in Python and Scheme: we want a base case and a recursive case.

We start by defining a local table called `num` that has 1 column `n`. Each row will have 1 value, a natural number. The base case is 0 (the smallest natural number) and we can create a one row table for it with `select 0`. For the recursive case, for each natural number, we can add 1 to get another natural number with `select n + 1 from num`. Since we want numbers up to 5, the recursive case applies only to numbers less than 5 (so  $n + 1$  will be 5 or less). Finally, we combine the two cases into a single table with `union`.

```
sqlite> create table naturals as
...>   with num(n) as (
...>     select 0 union
...>     select n + 1 from num where n < 5
...>   )
...>   select * from num;
```

Remember that `num` is local to this query. We can see the result by selecting everything from `naturals`.

```
sqlite> select * from naturals;
0
1
2
3
4
5
```

---

**5.3 Questions**

---

1. Write a query that creates a table called `factorials`, which has a single column, and a row for each factorial for the numbers 0 to 10 (inclusive). To do this, create a local table with two columns: the first column is the numbers 0 to 10 (inclusive), and the second column is their factorials.

```
create table factorials as
```

```
sqlite> select * from factorials;
```

```
1
1
2
6
24
120
720
5040
40320
362880
3628800
```

2. Write a query that groups natural numbers as sequences of 3's. Your solution should generate the following output.

```
0|1|2
3|4|5
6|7|8
9|10|11
12|13|14
```