DECLARATIVE PROGRAMMING 11

COMPUTER SCIENCE 61A

April 23, 2015

1 Declarative Programming

Over the semester, we have been using *imperative programming* – a programming style where code is written as a set of instructions for the computer. In this section, we introduce *declarative programming* – code that declares *what* we want, not *how* to compute it.

1.1 The SQL Language

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 design and perform a computational process to produce such a result.

A table, also called a relation, has a fixed number of named and typed columns. Each row of a table represents a data record and has one value for each] column. We can choose which columns to show, filter using a `where` clause, and sort with an `order by` clause using the following syntax:

```
select [columns] from [tables] where [condition] order by [order]
```

For example, we have a table named `records` that stores information about the employees at a small company[1]:

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
DeWitt Aull	Administration	Secretary	25000	Oliver Warbucks
Eben Scrooge	Accounting	Chief Accountant	75000	Oliver Warbucks
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

2 Creating Tables

We can use a `select` statement to create tables. We can define a new table either by listing the values in a single row and joining rows together 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.

```
sqlite> create table records as
   ...>     select "Ben Bitdiddle" as name, ...
   ...
```

Or, more commonly, project an existing table using a `from` clause, for example,

```
sqlite> select * from records where name = "Ben Bitdiddle";
Ben Bitdiddle|Computer|Wizard|60000|Oliver Warbucks
```

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
```

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

2.1 Questions

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

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

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

2.2 Joins

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

Division	Day	Time
Accounting	Monday	9am
Computer	Wednesday	4pm
Administration	Monday	11am
Administration	Thursday	1pm

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 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 $m \cdot n$ 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;
Alyssa P Hacker|Monday
...
Ben Bitdiddle|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
```

2.3 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.

2.4 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.

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

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

3 Local Tables

Select statements can optionally include a with clause that generates and names additional tables used in computing the final result. The full syntax of a select statement, not including unions, has the following form:

```
with [tbls] select [cols] from [nms] where [cond] order by [order]
```

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.

3.1 Recursion

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

Let's create a table of natural numbers from 0 to 3 (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`. `num` will take in 1 column `n`, which is the natural number. The base case will be 0 (the smallest natural number) and we can create a table for it with `select 0`. When we create the recursive case, we're always building 1 up from the previous natural number n (`select n + 1 from num`).

If we combine all these elements together, we get

```
sqlite> create table natural as
   ...>     with num(n) as (
   ...>         select 0 union
   ...>         select n + 1 from num
   ...>         where n < 3
   ...>     )
   ...>     select * from num;
```

Remember that `num` does not exist and if we want to find out what we made, we must ask for `natural`

```
sqlite> select * from natural;
0
1
2
3
```

3.2 Questions

1. Write a query that creates a table called `factorial`. To do so, first create a local table called `fact` with columns: `n`, being $n!$, and `nfact`, calculating out $n!$. Calculate the factorial from 0 through 10 (inclusive and ascending order).

2. Write a query that calculates the first 10 squares starting from 1.

3. 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
```

3.3 Extra Questions

1. Generate a table that contains all the permutations of length 3 from the set 0, 1, 2, 3. Your solution should have three columns named p, q, r. Hint: Use the `natural` table defined earlier.