CS 61A Spring 2018

SQL

Discussion 10: April 18, 2018 Solutions

1 Introduction

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.

`records`

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

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
```

[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 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
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
```

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, much like Scheme. 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)!

Questions

Our tables:

`records:` **Name** **Division** **Title** **Salary** **Supervisor**

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

```
SELECT name FROM records WHERE supervisor = "Oliver Warbucks";
```

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

```
SELECT * FROM records WHERE name = supervisor;
```

2.3 Write a query that outputs the names of all employees with salary greater than 50,000 in alphabetical order.

```
SELECT name FROM records WHERE salary > 50000 ORDER BY name;
```

3 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	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 (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
```

Questions

Our tables:

`records:` **Name** **Division** **Title** **Salary** **Supervisor**

`meetings:` **Division** **Day** **Time**

3.1 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.

```
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;
```

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

```
SELECT e.name FROM records AS e, records AS s
  WHERE e.supervisor = s.name AND e.division != s.division;
```

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

```
SELECT m.day, m.time FROM records AS r, meetings AS m
  WHERE r.division = m.division AND
        r.supervisor = "Oliver Warbucks";
```

3.4 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.

```
SELECT b.name FROM records AS a, records AS b
  WHERE a.supervisor = b.name AND b.supervisor != b.name;
```

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

```
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. If we wanted to remove duplicates, we could use the `DISTINCT` keyword.

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

```
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;
```

4 Modifying Tables

Tables don't need to begin fully formed, it's possible to update them after creation! We'll also introduce this alternative syntax for creating a table:

```
CREATE TABLE [table]([column1], [column2] DEFAULT [val], ...);
```

The **optional** `DEFAULT` keyword denotes default values for a given column if they're not specified. This will be relevant when we insert new elements into our table. To add a new table entries, use the `INSERT INTO` statement:

```
INSERT INTO [table] ([column1], [column2], ...)
  VALUES ([value1], [value2], ...), ([value1], [value2], ...);
```

A couple of notes:

- If a value is specified for each column of the table, you don't need to specify column names. This is because each value matches up with a column, so there's no ambiguity.
- For columns where a value is not specified, the default value will be used if available. If not, that column in the new row will be left empty!

Here's an example of insertion into an empty table:

```
sqlite> CREATE TABLE dogs(name, age, phrase DEFAULT "woof");
sqlite> INSERT INTO dogs(name, age) VALUES ("Fido", 1), ("Sparky", 2);
sqlite> INSERT INTO dogs VALUES ("Lassie", 2, "I'll save you!"), ("Floofy", 3);
Error: all VALUES must have the same number of terms
sqlite> INSERT INTO dogs VALUES ("Lassie", 2, "I'll save you!"), ("Floofy", 3, "Much doge");
sqlite> SELECT * FROM dogs;
Fido|1|woof
Sparky|2|woof
Lassie|2|I'll save you!
Floofy|3|Much doge
```

We can update certain entries in a table using `UPDATE`:

```
UPDATE [table] SET [column1] = [value1], [column2] = [value2], ... WHERE [condition];
```

All rows matching the condition will have their columns updated. If no condition is specified, **all** rows will be updated! We can also remove certain entries in a table using `DELETE`:

```
DELETE FROM [table] WHERE [condition];
```

Just like with `UPDATE`, if not condition is specified, **all** rows will be deleted! Here's an example using all of the above:

```
sqlite> UPDATE dogs SET age=age+1; -- If condition isn't specified, every row is updated
sqlite> SELECT * FROM dogs;
Fido|2|woof
Sparky|3|woof
Lassie|3|I'll save you!
Floofy|4|Much doge
```

```
sqlite> UPDATE dogs SET phrase = "bark" WHERE age=2;
sqlite> SELECT * FROM dogs WHERE age=2;
Fido|2|bark
sqlite> DELETE FROM dogs WHERE age=3;
sqlite> SELECT * FROM dogs;
Fido|2|bark
Floofy|4|Much doge
```

Finally, we can delete an entire table using the `DROP TABLE [table]` statement. In this example, the `.schema` statement shows us a list of the current tables, along with their column names.

```
sqlite> .schema
CREATE TABLE dogs(name, age, phrase DEFAULT "woof");
sqlite> DROP TABLE dogs;
sqlite> .schema
sqlite> -- Nothing displayed above
```

Questions

Our tables:

dogs: **Name** **Age** **Phrase, DEFAULT="woof"**

4.1 What would SQL display? **Keep track of the contents of the table after every statement below.**

```
sqlite> SELECT * FROM dogs;
Fido|1|woof
Sparky|2|woof
Lassie|2|I'll save you!
Floofy|3|Much doge
```

```
sqlite> INSERT INTO dogs(age, name) VALUES ("Rover", 3);
sqlite> SELECT * FROM dogs;
```

```
Fido|1|woof
Sparky|2|woof
Lassie|2|I'll save you!
Floofy|3|Much doge
3|Rover|woof
```

```
sqlite> UPDATE dogs SET name=age, age=name WHERE name=3;
sqlite> SELECT * FROM dogs;
```

```
Fido|1|woof
Sparky|2|woof
Lassie|2|I'll save you!
Floofy|3|Much doge
Rover|3|woof
```

```
sqlite> DELETE FROM dogs WHERE age < 3;
sqlite> SELECT * FROM dogs;
```

```
Floofy|3|Much doge
Rover|3|woof
```

5 Extra Questions

Use the following table called `courses` for the questions below:

`courses`

Professor	Course	Semester
John DeNero	CS 61A	Fa17
Paul Hilfinger	CS 61A	Fa17
Paul Hilfinger	CS 61A	Sp17
John DeNero	Data 8	Sp17
Josh Hug	CS 61B	Sp17
Satish Rao	CS 70	Sp17
Nicholas Weaver	CS 61C	Sp17
Gerald Friedland	CS 61C	Sp17
John DeNero	CS 61A	Fa16
Paul Hilfinger	CS 61B	Fa16
⋮	⋮	⋮

There's also an additional table which we have constructed from `courses`, named `num_taught`. It contains the number of times each professor has taught a given course:

`num_taught`

Professor	Course	Times
Gerald Friedland	CS 61C	1
John DeNero	CS 61A	2
John DeNero	Data 8	1
Josh Hug	CS 61B	1
Nicholas Weaver	CS 61C	1
Paul Hilfinger	CS 61A	2
Paul Hilfinger	CS 61B	1
Satish Rao	CS 70	1
⋮	⋮	⋮

5.1 Write a query that selects all courses Paul Hilfinger taught once.

```
SELECT course FROM num_taught WHERE professor = 'Paul Hilfinger' AND times = 1;
```

5.2 Write a query that outputs all the rows from `courses` where the professor teaching that semester has taught the course at least one other time.

```
SELECT a.professor, a.course, a.semester
  FROM courses AS a, num_taught as b
  WHERE a.professor = b.professor AND a.course = b.course AND b.times > 1;
```

5.3 Write a query that outputs two professors and a course if they have taught that course the same number of times.

```sql
SELECT a.professor, b.professor, a.course
  FROM num_taught AS a, num_taught AS b
  WHERE a.professor > b.professor
    AND a.course = b.course
    AND a.times = b.times;
```