SQL

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.

> For this discussion, you can test out your code at sql.cs61a.org. The **records** table should already be loaded in.

Select Statements

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
```

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
...
Robert Cratchet|Accounting
```

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

```
sqlite> SELECT * FROM records;
Alyssa P Hacker|Computer|Programmer|40000|Ben Bitdiddle
...
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 [columns] LIMIT [limit];
```

- `SELECT [columns]` tells SQL that we want to include the given columns in our output table; `[columns]` is a comma-separated list of column names, and `*` can be used to select all columns
- `FROM [tables]` tells SQL that the columns we want to select are from the given table(s)
- `WHERE [condition]` filters the output table by only including rows whose values satisfy the given `[condition]`, a boolean expression
- `ORDER BY [columns]` orders the rows in the output table by the given comma-separated list of columns
- `LIMIT [limit]` limits the number of rows in the output table by the integer `[limit]`

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 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, 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)!

SQL Queries

For the following questions, you will be referring to the **records** table:

Name	Division	Title	Salary	Supervisor
Alyssa P Hacker	Computer	Programmer	40000	Ben Bitdiddle
...	...	...	...	...
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

To see the full table, you can go to sql.cs61a.org and enter `SELECT * FROM records;`. However, it is not necessary to see the full table in order to complete the questions.

Q1: Oliver Employees

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

```
SELECT "YOUR CODE HERE" Name FROM records WHERE Supervisor = 'Oliver
Warbucks';
```

Q2: Self Supervisor

Write a query that outputs all information about employees that supervise themselves.

```
SELECT "YOUR CODE HERE" * FROM records WHERE Supervisor = Name;
```

Q3: Rich Employees

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

```
SELECT "YOUR CODE HERE" Name FROM records WHERE Salary > 50000
ORDER BY Name;
```

Note: This worksheet is a problem bank—most TAs will not cover all the problems in discussion section.

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	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 **m*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;
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
```

Join Questions

For the following questions, you will be referring to the **records** and **meetings** tables:

records

Name	Division	Title	Salary	Supervisor
Alyssa P Hacker	Computer	Programmer	40000	Ben Bitdiddle
...	...	...	...	...
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

meetings

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

Q4: Oliver Employee Meetings

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

```
SELECT "YOUR CODE HERE" Day, Time FROM records AS r, meetings AS s
    WHERE Supervisor = 'Oliver Warbucks' AND r.Division = s.Division;
```

```
# You can use more space on the back if you want
```

Q5: Different Division

name | division | supervisor | supervisor's division

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

```
SELECT "YOUR CODE HERE" r1.Name FROM records AS r1, records AS r2
    WHERE r1.Supervisor = r2.Name AND r1.Division != r2.Division;
```

Q6: Middle Manager

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 "YOUR CODE HERE" r1.Name FROM records AS r1, records AS r2
    WHERE r1.Name != r1.Supervisor AND
          r2.Supervisor = r1.Name;
```

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 minimum 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 rows, the HAVING clause filters out entire groups. 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
```

Aggregation Questions

For the following questions, you will be referring to the **records** and **meetings** tables:

records

Name	Division	Title	Salary	Supervisor
Alyssa P Hacker	Computer	Programmer	40000	Ben Bitdiddle
...	...	...	...	...
Robert Cratchet	Accounting	Scrivener	18000	Eben Scrooge

meetings

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

Q7: Supervisor Sum Salary

Write a query that outputs each supervisor and the sum of salaries of all the employees they supervise.

```
SELECT "YOUR CODE HERE" Supervisor, SUM(Salary) FROM records
GROUP BY Supervisor;
```

Q8: Num Meetings

Write a query that outputs the days of the week for which fewer than 5 employees have a meeting. You may assume no department has more than one meeting on a given day.

```
SELECT "YOUR CODE HERE" Day FROM records AS r, meetings AS m
WHERE r.Division = m.Division
GROUP BY Day HAVING COUNT(*) < 5;
```

← how to join properly?

Q9: Rich Pairs

Write a query that outputs all divisions for which there is more than one employee, and all pairs of employees within that division that have a combined salary less than 100,000.

```
SELECT "YOUR CODE HERE" r1.Division FROM records AS r1, records AS r2
WHERE r1.Division = r2.Division AND r1.Name < r2.Name
GROUP BY r1.Division HAVING MAX(r1.Salary + r2.Salary) < 1000;
# You can use more space on the back if you want
```

Div	S1	S2
A	100000	0
A	10	10
B	0	0
B	0	0

pw: division

~~(b)~~ (tree label (b))

(list _ _)