# STRUCTURED QUERY LANGUAGE (SQL)

COMPUTER SCIENCE MENTORS CS 88

April 26th - May 1st

## 1 Introduction to SQL

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 |

**Solution:** Video walkthrough

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`:

---

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

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

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
```

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)!

The GROUP BY and HAVING clauses of a SELECT statement are used to partition rows into groups and select only a subset of the groups. Any aggregate functions in the having clause or column description will apply to each group independently, rather than the entire set of rows in the table.

```
sqlite> SELECT division, max(salary) FROM records GROUP BY
   division;
Computer|60000
Administration|150000
Accounting|75000

sqlite> SELECT division, max(salary) FROM records GROUP BY
   division HAVING COUNT(*) > 1;
Computer|60000
Accounting|75000
```

```
                                       mentors
```

| Name | Food | Color | Editor | Language |
|---|---|---|---|---|
| Kaitlyn | Thai | Purple | Notepad++ | Java |
| Jessica | Pie | Green | Sublime | Java |
| Sohum | Sushi | Orange | Emacs | Ruby |
| Ada | Tacos | Blue | Vim | Python |
| Shreya | Ramen | Green | Vim | Python |

1. Write a query that lists all the mentors along with their favorite food if their favorite color is green.
   ```
   Jessica|Pie
   Shreya|Ramen
   ```

   > **Solution:**
   > ```
   > select name, food
   >   from mentors
   >   where color = 'Green';
   >
   > -- With aliasing
   > select m.name, m.food
   >   from mentors as m
   >   where m.color = 'Green';
   > ```

2. Write a query that lists the food and the color of every person whose favorite language is *not* Python.
   ```
   Sushi|Orange
   Pie|Green
   Thai|Purple
   ```

   > **Solution:**
   > ```
   > select food, color
   >   from mentors
   >   where language != 'Python';
   >
   > -- With aliasing
   > select m.food, m.color
   >   from mentors as m
   >   where m.language <> 'Python';
   > ```

3. Write a query that lists all the pairs of mentors who like the same language. (How can we make sure to remove duplicates?)
```
Kaitlyn | Jessica
Shreya | Ada
```

> **Solution:**
> ```
> select m1.name, m2.name
>     from mentors as m1, mentors as m2
>     where m1.language = m2.language and m1.name > m2.name;
> ```

4. Write a query that has the same data, but alphabetizes the rows by name. (Hint: Use `order by`.)
```
Ada|Tacos|Blue|Vim|Python
Jessica|Pie|Green|Sublime|Java
Kaitlyn|Thai|Purple|Notepad++|Java
Shreya|Ramen|Green|Vim|Python
Sohum|Sushi|Orange|Emacs|Ruby
```

> **Solution:**
> ```
> select * from mentors order by name;
> ```

## 2   Aggregation

CS 88 wants to start a fish hatchery, and we need your help to analyze the data Stephen has collected for the fish populations! Running a hatchery is expensive – we'd like to make some money on the side by selling some seafood (only older fish of course) to make delicious sushi.

The table `fish` contains a subset of the data that has been collected. The SQL column names are listed in brackets.

| Species [species] | Population [pop] | Breeding Rate [rate] | $/piece [price] | # of pieces per fish [pieces] |
|---|---|---|---|---|
| Salmon | 500 | 3.3 | 4 | 30 |
| Eel | 100 | 1.3 | 4 | 15 |
| Yellowtail | 700 | 2.0 | 3 | 30 |
| Tuna | 600 | 1.1 | 3 | 20 |

5. Write a query to find the three most populated fish species.

> **Solution:**
> ```
> select species from fish order by -pop LIMIT 3;
> ```

6. Write a query to find the total number of fish in the ocean. Additionally, include the number of species we summed. Your output should have the number of species and the total population.

> **Solution:**
> ```
> select COUNT(species), SUM(pop) from fish;
> ```

7. Now, as a customer, you want to get the best deal possible. Write a query to select the most number of pieces for each price. Your output should include the price and pieces. (Hint: use `group by`.)

> **Solution:**
> ```
> select price, MAX(pieces) from fish GROUP BY price;
> ```

## 3  Extra Problems

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

<div align="center">

courses

| Professor | Course | Semester |
|-----------|--------|----------|
| Dan Garcia | CS 61C | Sp19 |
| John DeNero | CS 61A | Fa18 |
| Dan Garcia | CS 10 | Fa18 |
| Josh Hug | CS 61B | Sp18 |
| John DeNero | CS 61A | Sp18 |
| 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 |
| ⋮ | ⋮ | ⋮ |

</div>

8. Create a table called `num_taught` that contains three columns: `professor`, the `course` they taught, and the number of `times` they taught each course.
   ```
   CREATE TABLE num_taught AS
   SELECT
   ```

   > **Solution:**
   > ```
   > SELECT professor AS professor, course AS course, COUNT(*)
   >     AS times
   > FROM courses GROUP BY professor, course;
   > ```

9. Write a query that outputs two professors and a course if they have taught that course the same number of times. You may use the `num_taught` table you created in the previous question.

   > **Solution:**
   > ```
   > 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;
   > ```

☐

10. Write a query that outputs two professors if they co-taught (taught the same course at the same time) the same course more than once.

> **Solution:**
> ```
> SELECT a.professor, b.professor
>   FROM courses AS a, courses AS b
>   WHERE a.professor < b.professor AND
>     a.semester = b.semester and a.course = b.course
>   GROUP BY a.course, a.professor, b.professor HAVING COUNT
>     (*) > 1;
> ```