SQL AND FINAL REVIEW

CS 61A GROUP MENTORING

April 23 to April 25, 2018

1 Creating Tables, Querying Data

Examine the table, `mentors`, depicted below.

Name	Food	Color	Editor	Language
Jacob	Thai	Purple	Notepad++	Java
Rachel	Pie	Green	Sublime	Java
Jemmy	Sushi	Orange	Emacs	Ruby
Daniel	Tacos	Blue	Vim	Python
Amy	Ramen	Green	Vim	Python

1. Create a new table `mentors` that contains all the information above. (You only have to write out the first two rows.)

Solution:
```
create table mentors as
  select 'Jacob' as name, 'Thai' as food, 'Purple' as
     color, 'Notepad++' as editor, 'Java' as language union
  select 'Rachel', 'Pie', 'Green', 'Sublime', 'Java' union
  select 'Jemmy', 'Sushi', 'Orange', 'Emacs', 'Ruby' union
  select 'Daniel', 'Tacos', 'Blue', 'Vim', 'Python' union
  select 'Amy', 'Ramen', 'Green', 'Vim', 'Python';
```

2. Write a query that lists all the mentors along with their favorite food if their favorite color is green.

```
Rachel|Pie
Amy|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';
```

3. 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';
```

4. Write a query that lists all the pairs of mentors who like the same language. (How can we make sure to remove duplicates?)

```
Rachel|Jacob
Daniel|Amy
```

Solution:

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

2 Aggregation

CS 61A wants to start a fish hatchery, and we need your help to analyze the data we've 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

Hint: The aggregate functions `MAX`, `MIN`, `COUNT`, and `SUM` return the maximum, minimum, number, and sum of the values in a column. The `GROUP BY` clause of a select statement is used to partition rows into groups.

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

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

2. 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;
```

3. Profit is good, but more profit is better. Write a query to select the species that yields the most number of pieces for each price. Your output should include the species, price, and pieces.

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

The table `competitor` contains the competitor's `price` for each `species`.

Species [species]	**$/piece [price]**
Salmon	2
Eel	3.4
Yellowtail	3.2
Tuna	2.6

4. Business is good, but a bunch of competition has sprung up! Through some cunning corporate espionage, we have determined one such competitor's selling prices.

   Write a query that returns, for each species, the difference between our hatchery's revenue versus the competitor's revenue for one whole fish. For example, the table should contain the following row `Salmon|60`.

   Because we make 30 pieces at $4 a piece for $120, whereas the competitor will make 30 pieces at $2 a piece for $60. Finally, the difference is 60.

Solution:
```
select fish.species, (fish.price - competitor.price) *
   pieces
    from fish, competitor
    where fish.species = competitor.species;
```

5. Suppose these fish breed every day. The population of each fish gets multiplied by its breeding rate every year. Write a series of SQL statements which will create a table containing each species of fish, their population, and the year for this year (year 0), next year, and the year after. You have access to all tables defined in previous parts.
   `CREATE TABLE yearly_pop(species, pop, n);`

   Solution:
   ```
   INSERT INTO yearly_pop SELECT species, pop, 0 FROM fish;

   INSERT INTO yearly_pop SELECT yearly_pop.species,
      yearly_pop.pop * rate, n + 1
                                       FROM fish, yearly_pop
                                          WHERE
                                          yearly_pop.species
                                          = fish.species;

   INSERT INTO yearly_pop SELECT yearly_pop.species,
      yearly_pop.pop * rate, n + 1
                                       FROM fish, yearly_pop
                                          WHERE
                                          yearly_pop.species
                                          = fish.species and
                                          n = 1;
   ```

6. Mutate the same table to only include rows for year 2.

   Solution:
   ```
   DELETE FROM yearly_pop WHERE n <> 2;
   ```

7. Tragedy strikes! In year 2, all fish with a population less than 1,000 die, so remove them from the table. In addition, the remaining fish have their population halved, so mutate the table for them similarly.

   Solution:
   ```
   DELETE FROM yearly_pop WHERE pop < 1000;
   UPDATE yearly_pop set pop = pop * .5
   ```