Aggregation

Objectives

- Define "aggregation" and give examples of its use.
- Write queries that compute aggregated values.
- Trace the execution of a query that performs aggregation.
- Explain how missing data is handled during aggregation.

We now want to calculate ranges and averages for our data. We know how to select all of the dates from the `Visited` table:

```
%load_ext sqlitemagic
```

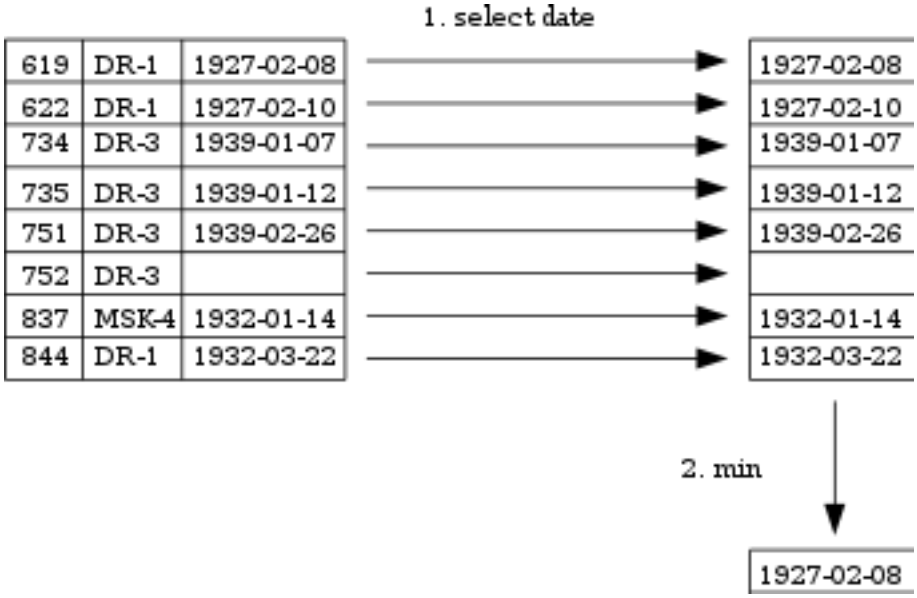
```
%%sqlite survey.db
select dated from Visited;
```

```
1927-02-08
1927-02-10
1939-01-07
1930-01-12
1930-02-26
None
1932-01-14
1932-03-22
```

but to combine them, wee must use an aggregation function (../../gloss.html#aggregation-function) such as `min` or `max`. Each of these functions takes a set of records as input, and produces a single record as output:

```
%%sqlite survey.db
select min(dated) from Visited;
```

```
1927-02-08
```




```
%%sqlite survey.db
select max(dated) from Visited;
```

```
1939-01-07
```

`min` and `max` are just two of the aggregation functions built into SQL. Three others are `avg`, `count`, and `sum`:

```
%%sqlite survey.db
select avg(reading) from Survey where quant='sal';
```

```
7.20333333333
```

```
%%sqlite survey.db
select count(reading) from Survey where quant='sal';
```

```
9
```

```
%%sqlite survey.db
select sum(reading) from Survey where quant='sal';
```

```
64.83
```

We used `count(reading)` here, but we could just as easily have counted `quant` or any other field in the table, or even used `count(*)`, since the function doesn't care about the values themselves, just how many values there are.

SQL lets us do several aggregations at once. We can, for example, find the range of sensible salinity measurements:

```
%%sqlite survey.db
select min(reading), max(reading) from Survey where quant='sal' and r
eading<=1.0;
```

```
0.050.21
```

We can also combine aggregated results with raw results, although the output might surprise you:

```
%%sqlite survey.db
select person, count(*) from Survey where quant='sal' and reading<=1.
0;
```

```
lake7
```

Why does Lake's name appear rather than Roerich's or Dyer's? The answer is that when it has to aggregate a field, but isn't told how to, the database manager chooses an actual value from the input set. It might use the first one processed, the last one, or something else entirely.

Another important fact is that when there are no values to aggregate, aggregation's result is "don't know" rather than zero or some other arbitrary value:

```
%%sqlite survey.db
select person, max(reading), sum(reading) from Survey where quant='mi
ssing';
```

```
NoneNoneNone
```

One final important feature of aggregation functions is that they are inconsistent with the rest of SQL in a very useful way. If we add two values, and one of them is null, the result is null. By extension, if we use `sum` to add all the values in a set, and any of those values are null, the result should also be null. It's much more useful, though, for aggregation functions to ignore null values and only combine those that are non-null. This behavior lets us write our queries as:

```
%%sqlite survey.db
select min(dated) from Visited;
```

```
1927-02-08
```

instead of always having to filter explicitly:

```
%%sqlite survey.db
select min(dated) from Visited where dated is not null;
```

```
1927-02-08
```

Aggregating all records at once doesn't always make sense. For example, suppose Gina

suspects that there is a systematic bias in her data, and that some scientists' radiation readings are higher than others. We know that this doesn't work:

```
%%sqlite survey.db
select person, count(reading), round(avg(reading), 2)
from  Survey
where quant='rad';
```

```
roe 8 6.56
```

because the database manager selects a single arbitrary scientist's name rather than aggregating separately for each scientist. Since there are only five scientists, she could write five queries of the form:

```
%%sqlite survey.db
select person, count(reading), round(avg(reading), 2)
from  Survey
where quant='rad'
and   person='dyer';
```

```
dyer 2 8.81
```

but this would be tedious, and if she ever had a data set with fifty or five hundred scientists, the chances of her getting all of those queries right is small.

What we need to do is tell the database manager to aggregate the hours for each scientist separately using a `group by` clause:

```
%%sqlite survey.db
select   person, count(reading), round(avg(reading), 2)
from     Survey
where    quant='rad'
group by person;
```

```
dyer 2 8.81
lake 2 1.82
pb   3 6.66
roe  1 11.25
```

`group by` does exactly what its name implies: groups all the records with the same value for the specified field together so that aggregation can process each batch separately. Since all the records in each batch have the same value for `person`, it no longer matters that the database manager is picking an arbitrary one to display alongside the aggregated `reading` values.

Just as we can sort by multiple criteria at once, we can also group by multiple criteria. To get the average reading by scientist and quantity measured, for example, we just add another field to the `group by` clause:

```
%%sqlite survey.db
select   person, quant, count(reading), round(avg(reading), 2)
from     Survey
group by person, quant;
```

```
None sal  1 0.06
None temp 1 -26.0
dyer rad  2 8.81
dyer sal  2 0.11
lake rad  2 1.82
lake sal  4 0.11
lake temp 1 -16.0
pb   rad  3 6.66
pb   temp 2 -20.0
roe  rad  1 11.25
roe  sal  2 32.05
```

Note that we have added `person` to the list of fields displayed, since the results wouldn't make much sense otherwise.

Let's go one step further and remove all the entries where we don't know who took the measurement:

```
%%sqlite survey.db
select   person, quant, count(reading), round(avg(reading), 2)
from     Survey
where    person is not null
group by person, quant
order by person, quant;
```

```
dyer rad  2 8.81
dyer sal  2 0.11
lake rad  2 1.82
lake sal  4 0.11
lake temp 1 -16.0
pb   rad  3 6.66
pb   temp 2 -20.0
roe  rad  1 11.25
roe  sal  2 32.05
```

Looking more closely, this query:

1. selected records from the `Survey` table where the `person` field was not null;
2. grouped those records into subsets so that the `person` and `quant` values in each subset were the same;
3. ordered those subsets first by `person`, and then within each sub-group by `quant`; and

4. counted the number of records in each subset, calculated the average `reading` in each, and chose a `person` and `quant` value from each (it doesn't matter which ones, since they're all equal).

Challenges

1. How many temperature readings did Frank Pabodie record, and what was their average value?
2. The average of a set of values is the sum of the values divided by the number of values. Does this mean that the `avg` function returns 2.0 or 3.0 when given the values 1.0, `null`, and 5.0?
3. We want to calculate the difference between each individual radiation reading and the average of all the radiation readings. We write the query:

   ```
   select reading - avg(reading) from Survey where quant='rad';
   ```

   What does this actually produce, and why?
4. The function `group_concat(field, separator)` concatenates all the values in a field using the specified separator character (or ',' if the separator isn't specified). Use this to produce a one-line list of scientists' names, such as:

   ```
   William Dyer, Frank Pabodie, Anderson Lake, Valentina Roerich, Frank Danfort
   h
   ```

   Can you find a way to order the list by surname?

Key Points

- An aggregation function combines many values to produce a single new value.
- Aggregation functions ignore `null` values.
- Aggregation happens after filtering.
