software carpentry (http://software-carpentry.org)

Missing Data

Objectives

- Explain how databases represent missing information.
- Explain the three-valued logic databases use when manipulating missing information.
- Write queries that handle missing information correctly.

Real-world data is never complete—there are always holes. Databases represent these holes using special value called `null`. `null` is not zero, `False`, or the empty string; it is a one-of-a-kind value that means "nothing here". Dealing with `null` requires a few special tricks and some careful thinking.

To start, let's have a look at the `Visited` table. There are eight records, but #752 doesn't have a date—or rather, its date is null:

```
%load_ext sqlitemagic
```

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

```
619 DR-1  1927-02-08
622 DR-1  1927-02-10
734 DR-3  1939-01-07
735 DR-3  1930-01-12
751 DR-3  1930-02-26
752 DR-3  None
837 MSK-4 1932-01-14
844 DR-1  1932-03-22
```

Null doesn't behave like other values. If we select the records that come before 1930:

```
%%sqlite survey.db
select * from Visited where dated<'1930-00-00';
```

```
619 DR-1 1927-02-08
622 DR-1 1927-02-10
```

we get two results, and if we select the ones that come during or after 1930:

```
%%sqlite survey.db
select * from Visited where dated>='1930-00-00';
```

```
734 DR-3  1939-01-07
735 DR-3  1930-01-12
751 DR-3  1930-02-26
837 MSK-4 1932-01-14
844 DR-1  1932-03-22
```

we get five, but record #752 isn't in either set of results. The reason is that `null<'1930-00-00'` is neither true nor false: null means, "We don't know," and if we don't know the value on the left side of a comparison, we don't know whether the comparison is true or false. Since databases represent "don't know" as null, the value of `null<'1930-00-00'` is actually `null`. `null>='1930-00-00'` is also null because we can't answer to that question either. And since the only records kept by a `where` are those for which the test is true, record #752 isn't included in either set of results.

Comparisons aren't the only operations that behave this way with nulls. `1+null` is `null`, `5*null` is `null`, `log(null)` is `null`, and so on. In particular, comparing things to null with = and != produces null:

```
%%sqlite survey.db
select * from Visited where dated=NULL;
```

```
```

```
%%sqlite survey.db
select * from Visited where dated!=NULL;
```

```
```

To check whether a value is `null` or not, we must use a special test `is null`:

```
%%sqlite survey.db
select * from Visited where dated is NULL;
```

```
752 DR-3 None
```

or its inverse `is not null`:

```
%%sqlite survey.db
select * from Visited where dated is not NULL;
```

```
619 DR-1  1927-02-08
622 DR-1  1927-02-10
734 DR-3  1939-01-07
735 DR-3  1930-01-12
751 DR-3  1930-02-26
837 MSK-4 1932-01-14
844 DR-1  1932-03-22
```

Null values cause headaches wherever they appear. For example, suppose we want to find all the salinity measurements that weren't taken by Dyer. It's natural to write the query like this:

```
%%sqlite survey.db
select * from Survey where quant='sal' and person!='lake';
```

```
619 dyer sal 0.13
622 dyer sal 0.09
752 roe  sal 41.6
837 roe  sal 22.5
```

but this query filters omits the records where we don't know who took the measurement. Once again, the reason is that when `person` is `null`, the `!=` comparison produces `null`, so the record isn't kept in our results. If we want to keep these records we need to add an explicit check:

```
%%sqlite survey.db
select * from Survey where quant='sal' and (person!='lake' or person
is null);
```

```
619 dyer sal 0.13
622 dyer sal 0.09
735 None sal 0.06
752 roe  sal 41.6
837 roe  sal 22.5
```

We still have to decide whether this is the right thing to do or not. If we want to be absolutely sure that we aren't including any measurements by Lake in our results, we need to exclude all the records for which we don't know who did the work.

Challenges

1. Write a query that sorts the records in `Visited` by date, omitting entries for which the date is not known (i.e., is null).

2. What do you expect the query:

   ```
   select * from Visited where dated in ('1927-02-08', null);
   ```

   to produce? What does it actually produce?

3. Some database designers prefer to use a sentinel value (../../gloss.html#sentinel-value) to mark missing data rather than `null`. For example, they will use the date "0000-00-00" to mark a missing date, or -1.0 to mark a missing salinity or radiation reading (since actual readings cannot be negative). What does this simplify? What burdens or risks does it introduce?

Key Points

- Databases use `null` to represent missing information.
- Any arithmetic or Boolean operation involving `null` produces `null` as a result.
- The only operators that can safely be used with `null` are `is null` and `is not null`.
