Currently, when a Date and DateTime are compared, the Date is treated as a DateTime with time "00:00:00 +0000". This results in behavior which could be surprising when the other DateTime is not in UTC (+0000):

```ruby
date = Date.new(2020, 1, 1) # 2020-01-01
datetime1 = DateTime.new(2020, 1, 1, 0, 0, 0, "+0100") # 2020-01-01 00:00:00 +0100
datetime2 = DateTime.new(2020, 1, 1, 1, 0, 0, "+0100") # 2020-01-01 01:00:01 +0100

date <=> datetime1 # 2020-01-01 00:00:00 +0000 <=> 2020-01-01 00:00:00 +0100
# == 1

date <=> datetime2 # 2020-01-01 00:00:00 +0000 <=> 2020-01-01 01:00:01 +0100
# == -1
```

I think it would be less surprising if the comparison used the offset of the other DateTime:

```ruby

date <=> datetime1 # 2020-01-01 00:00:00 +0100 <=> 2020-01-01 00:00:00 +0100
# == 0

date <=> datetime2 # 2020-01-01 00:00:00 +0100 <=> 2020-01-01 01:00:01 +0100
# == -1
```

Or, another possibility is to go one step further and convert the other DateTime to Date:

```ruby

date <=> datetime1 # 2020-01-01 00:00:00 +0100 <=> 2020-01-01 00:00:00 +0100
# == 0

date <=> datetime2 # 2020-01-01 00:00:00 +0100 <=> 2020-01-01 01:00:01 +0100
# == 0
```

This last behavior leans on the idea that a Date is semantically a range of Datetimes, instead of a single DateTime with its time omitted. It would also cause (date <=> datetime) == 0 when (date === datetime) == true.

Of course, these changes may break backwards compatibility, and so they may not be feasible. But, related issues have been discussed in rails/rails#36462 and rails/rails#36579, and fixing this behavior in Ruby itself would solve them.