Avoiding race conditions using `F()`

Another useful benefit of `F()` is that having the database - rather than Python - update a field's value avoids a *race condition*.

If two Python threads execute the code in the first example above, one thread could retrieve, increment, and save a field's value after the other has retrieved it from the database. The value that the second thread saves will be based on the original value; the work of the first thread will be lost.

If the database is responsible for updating the field, the process is more robust: it will only ever update the field based on the value of the field in the database when the `save()` or `update()` is executed, rather than based on its value when the instance was retrieved.

`F()` assignments persist after `Model.save()` ¶

`F()` objects assigned to model fields persist after saving the model instance and will be applied on each `save()`. For example:

```
reporter = Reporters.objects.get(name='Tintin')
reporter.stories_filed = F('stories_filed') + 1
reporter.save()

reporter.name = 'Tintin Jr.'
reporter.save()
```

`stories_filed` will be updated twice in this case. If it's initially `1`, the final value will be `3`. This persistence can be avoided by reloading the model object after saving it, for example, by using `refresh_from_db()`.
