

# Re: What gives data meaning?

---

*Source:* <http://coding.derkeiler.com/Archive/General/comp.object/2007-04/msg00055.html>

---

- *From:* "frebe" <[frebe73@xxxxxxxxxx](mailto:frebe73@xxxxxxxxxx)>
  - *Date:* 4 Apr 2007 10:47:47 -0700
- 

But to the point, if a program was able to store improperly-formatted zipcode inside the DB then whose fault is that?

what's an improperly formatted zipcode? In the US, you have 5 digits, in the netherlands you have 4 digits and 2 characters. A zipcode of 1234 AA is properly formatted for a dutch user of the application, but not correctly formatted for the US user of the program. Hence: context.

This thus means that if the db stores '1234AA', it can do so, and the dutch user will happily use it. The US user can't because for the US user it's just data, 1234 and 2 characters, it's not information (zipcode).

If you think about it, neither zip code means anything to the database. They're simply characters in a field.

With a proper type system or check constraints, a RDBMS could guarantee that no invalid zip codes are stored into the database. If we want to store zip codes for multiple countries, one solution is to have different relations for different countries with different domains/types for the zip code attributes.

```
address(id, country, street, ...)
dutch_address(id, country, dutch_zipcode)
us_address(id, country, us_zipcode)
```

The country columns in `dutch_address` and `us_address` should be constants (enforced by a check constraint), and there should be two foreign keys between `dutch_address/us_address (id, country)` referencing `address(id, country)`. Doing this, it would not be possible to have both a dutch and us address. The format of `dutch_zipcode` and `us_zipcode` could be enforced by a check constraint using regular expressions, or by

## Re: What gives data meaning?

defining a custom type.

To make it simpler to use, we would create a view like this:

```
create view address_all
select id, country, street, dutch_zipcode as zip
from address a join dutch_address da on da.id=a.id
union
select id, country, street, us_zipcode
from address a join us_address ua on ua.id=a.id
```

So that takes us back to whether or not the database is correct beyond its integrity checks. If it stores exactly what it should store then it is correct. An incorrect or improperly formatted zip code isn't the database's responsibility if that's what a human or application told it to store.

A RDBMS is capable of enforcing such constraints. We want all data in the database to be valid.

This reinforces the DB can be responsible for structural, type, and referential integrity, but it can not give meaning to its data.

A RDBMS can give meaning to its data in the same way as an application can give "meaning" to its data.

So this takes us back to our responsibility as programmers and designers to guard our database's integrity.

I am afraid that you say that "programmers" should guard the database because you want to use stored procedures as proxies to the database. The data integrity is enforced by primary and foreign key constraints, check constraints and triggers, in that preferred order. Hiding the database behind procedures is almost never necessary.

/Fredrik

.