Perl e ORM(e) su PostgreSQL

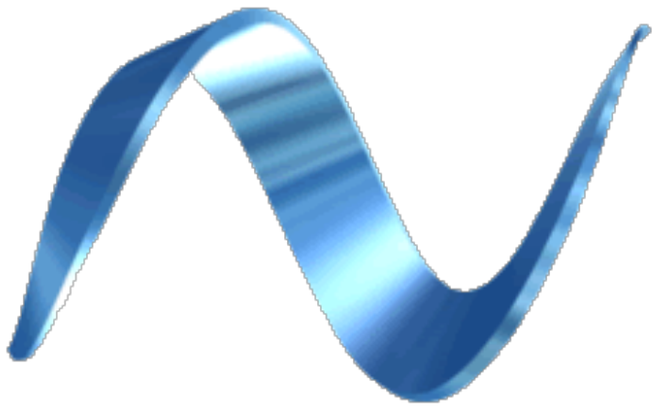
PgDay 2011 - Prato

by

Ferruccio Zamuner

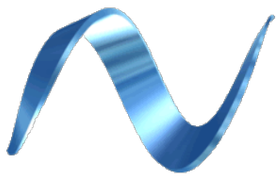
NONSOLOSOFT

Soluzioni per l'e-business



nonsolosoft@diff.org
http://www.nonsolosoft.com/

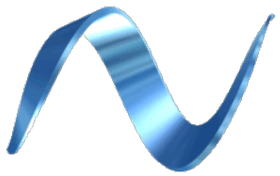
NONSOLOSOFT
Soluzioni per l'e-business



Perl

- linguaggio di scripting
- 25 anni di storia
- vari paradigmi di programmazione
- ha avuto successo in vari ambiti

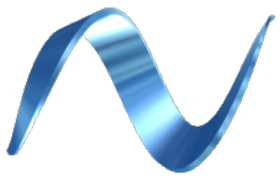
NONSOLOSOFT
Soluzioni per l'e-business



Perl e integrazione

- attitudine a dialogare con altri linguaggi
- multipiattaforma
- dotato della più ampia biblioteca di moduli
- dialoga con quasi tutti i db

NONSOLOSOFT
Soluzioni per l'e-business



Object Relational Mapping

è ormai un componente software presente in tutti i linguaggi

Design Pattern

ORM - Object relational Mapping

Facilita la progettazione e scrittura
Domain Driven Design,
ma si scosta
da chi è Relational Database Design centrico:
il database diventa uno storage
per gli stati e per i grafi degli oggetti dichiarati nell'ORM.

incapsulamento

ORM mira a nascondere l'implementazione
di come gli stati sono memorizzati
nello storage (RDBMS) avendo dei metodi che agiscono
sugli stati per leggerli e alterarli.

Relational Database Design

Mentre lo scopo del RDD e' quello di fornire un rapido e funzionale accesso ai dati stessi, riconoscendo le relazioni che intercorrono tra i tipi di dati (entita') e espondendole (operazione inversa all'incapsulamento degli ORM).

Pro ORM

penso ad oggetti

ho un controllo della sincronia delle operazioni

gli attributi del record acquisiscono propri metodi di accesso
che posso distinguere per sola lettura o meno

le chiavi esterne acquisiscono a loro volta metodi di accesso
alle relazioni esterne

$nodo->successivo->successivo->nome

la business logic e' facilmente esprimibile nel linguaggio di
programmazione
preferito

debugging piu' comodo che nel linguaggi procedurali nel db

Contro ORM

perdo il controllo di come sono eseguite le query SQL, in quale ordine

limiti sulla sintassi disponibile

se abbiamo un'equivalenza tra il peso di accesso ad un singolo record
e alle informazioni sulle relazioni 1:1 che esso ha, ottenere un elenco
puo' generare una select per record nel momento in cui uso gli accessor

la sintassi di interrogazione diventa un nuovo linguaggio

la logica che scrivo regola il funzionamento e i vincoli sui db,
devo esporre API per permettere
ad altri programmi di usufruire di questa logica
ma non posso impedire futuri accessi diretti ai dati

SQL invece...

parto dai dati

posso spingere nel db parti di logica che semplificando l'applicazione e migliorano le prestazioni (fino a quando sono sul singolo sql server)

la logica inserita a livello db permette l'accesso ai dati da diverse applicazioni
e i vincoli sono applicati sempre

DBIx::Class

Ci sono due modi di procedere:
1) scrivo DML in SQL per poi generare le classi

```
dbicdump -o dump_directory=./lib \
        -o components='["InflateColumn::DateTime"]' \
        MyApp::Schema 'dbi:pg:dname=sbo_devel' \
        user pass
```

2) scrivo le classi Perl con cui generare poi le tabelle

```
dbicdeploy MyApp::Schema DBI:Pg:foo \
        user pass '{ pg_enable_utf8 => 1 }'
```

```
package My::Schema::User;

...

__PACKAGE__->resultset_class(
  'My::ResultSet::User'
);
```

Attributi:

```
__PACKAGE__->add_columns(
 "id",
 { data_type => "integer", is_nullable => 0 },
 "username",
 { data_type => "text", is_nullable => 1 },
);
```

insert

```
  my $rs = $schema->resultset('CD');

  my $cd= $rs->create({title=>'postgresql is my
db',year=>'1997'})

  $rs->populate([
    { title => 'My parents sold me to a record company',
      year => 2005 },
    { title => 'Why Am I So Ugly?',
      year => 2006 },
    { title => 'I Got Surgery and am now Popular',
      year => 2007 }
]);
```

delete

```
$schema->resultset('Picture')
    ->search({ id => { -in => \@ids } })
    ->increment_views;
```

```
$rs->delete;
```

Vincoli

```
__PACKAGE__->set_primary_key("id");
```

Relazioni - Chiavi esterne

```
__PACKAGE__->belongs_to(
 "successivo",
 "PgD::Pg::Crediti",
 { "foreign.id" => "self.consolidato" },
);

__PACKAGE__->belongs_to(
 "credito",
 "PgD::Pg::Crediti",
 { "foreign.id" => "self.id" },
 { cascade_copy => 0, cascade_delete => 1 },
);
```

accessor

```
my $user = $schema->resultset('Users')->
                    find(userid=>$uid);
print $user->nome;    # Mario
print $user->profilo->website;
```

relazioni 1:N

```
my $user = $schema->resultset('Users')->
                    find(userid=>$uid);
$rs = $user->search_related('documents');

for my $doc ($rs->next) {
        print $doc->title;
}
```

viste

```
__PACKAGE__->table_class('DBIx::Class::ResultSource::View');

__PACKAGE__->table("albero_crediti");

/* 1 virtuale 0 fisica */

__PACKAGE__->result_source_instance->is_virtual(1);
__PACKAGE__->result_source_instance->view_definition(
   "with recursive t(node,consolidato,path) as
  (select id, consolidato, array[id]
     from crediti
    where id = ?
    union all
    select cr.id, cr.consolidato, t.path || array[cr.id]
      from crediti as cr
           join t on (cr.id = t.consolidato OR t.node=cr.consolidato)
    where not cr.id = any(path)
    )
select
 t.node as id, consolidato
 from t"
        );
```

Grazie!

domande?

per saperne di più:

http://www.dbix-class.org/documentation.html
http://www.cpan.org/
http://www.catalystframework.org/