DBIx::DataModel

Classes and UML-style associations on top of DBI

(just an appetizer...)

laurent.dami@justice.ge.ch
Agenda

- Introduction
  - ORMs
  - Design issues
- Unified Modelling Language (UML)
- Tables
  - Declaration
  - Usage
- Associations
  - Declaration
  - Usage
Introduction

◆ ORMs
◆ Design issues
ORM: Object-Relational Mapping

**RAM**
- r1: class1
  - +c1
  - +c2
  - +c3
- r2: class1

**DBMS**
- table1
  - c1 c2 c3
  - r1
  - r2
  - ...
- table2
  - c3 c4
  - ...

**Meaning:**
- r1 and r2 are instances of class1.
- The class1 objects are mapped to the tables table1 and table2 in the DBMS.
- The columns c1, c2, c3, c4 are present in the tables.
- The mapping between RAM and DBMS is illustrated through the relations r1 and r2.
What for?

[catalyst list] On Thu, 2006-06-08, Steve wrote:

- Not intending to start any sort of rancorous discussion,
- but I was wondering whether someone could illuminate me a little?

- I'm comfortable with SQL, and with DBI. I write basic SQL that runs just fine on all databases, or more complex SQL when I want to target a single database (usually postgresql).

- What value does an ORM add for a user like me?
ORM useful for ...

- navigation between tables
- data conversions
- generate complex SQL queries from Perl datastructures
- expansion of tree data structures coded in the relational model
- transaction encapsulation
- data validation
- auto-filling some columns at update

See Also: http://lists.rawmode.org/pipermail/catalyst/2006-June
ORMs in CPAN : TIMTOWTDI !

... DBIx::SQLEngine ... DBIx::RecordSet
Class::DBI
Tangram
Rose::DB::Object
DBIx::Class
ORM

DBIx::DataModel
Class::PObj ect
Alzabo
SPOPS
Some design issues

- DRY: Don't Repeat Yourself
- When to fetch column values, and which
- Pure objects versus Perl datastructures
- Encapsulation / collaboration with lower-level layers
- Fine SQL tuning
- Caching
Unified Modeling Language
UML conceptual Model: example
UML terminology

- **class**
  - Author
    - 1
    - *
  - Distribution
    - 1
    - *

- **association**
  - multiplicity
  - contains
    - contains
      - depends on

- **role name**
  - Package
    - parent
      - *
    - child
      - *
  - Module
    - *

Model implementation

Author
- author_id
- author_name
- e_mail

Distribution
- distrib_id
- distrib_name
- d_release

Dependency
- distrib_id
- module_id

Module
- module_id
- module_name

link table for n-to-n association
Tables [ and Views ]

- declaration
- usage
Schema and table declarations

# declare schema (Module-Author-Distribution)
DBIx::DataModel->Schema('MAD');

# declare tables in schema

# Class  DB_table  Primary_key
MAD->Table([qw/Author  t_author  author_id /]);
MAD->Table([qw/Distrib  t_distrib  distrib_id/]);
MAD->Table([qw/Module  t_module  module_id /]);
Don't Repeat Yourself

generate Data Definition Statements

generate Data Definition Statements

generate Data Definition Statements

get metadata from DBMS schema

get metadata from DBMS schema

get metadata from DBMS schema

DBMS

DBMS

DBMS

Prog. Lang.

Prog. Lang.

Prog. Lang.

DBMS

DBMS

DBMS
Design philosophy

Perl idioms: dual nature of Perl objects, dynamic typing, multiple inheritance, etc.

DBMS tools for declaring schema, datatypes, integrity rules, etc.

keep intersection to a strict minimum

freedom

responsability
Fetching data : example 1

# fetch from primary key
# by default, retrieves all columns ('*')
my $author = Author->fetch('dami');

# reach columns through the hashref API ...
while (my ($k, $v) = each %$author) {
    print "$k : $v\n";
}

# ... or use object-oriented methods
MAD->Autoload(1);  # Autoload is off by default
print $author->e_mail();
Fetching data: example 2

# select multiple records
my $recent_distribs = Distrib->select(
    -columns => [qw/distrib_name d_release/],
    -where => {d_release => {'>' => $some_date}},
    -orderBy => 'd_release DESC',
);

foreach my $distrib (@$recent_distribs) {...}
Select API

TableOrView->select(
    -columns => \@columns,
    # OR: -distinct => \@columns,
    -where => \%where,
    -groupBy => \@groupings,
    -having => \%criteria,
    -orderBy => \@order,
    -for => 'read only',
    -preExec => \&preExec_callback,
    -postExec => \&preExec_callback,
    -resultAs => 'rows' || 'sth' || 'sql'
        || 'iterator');

See Also: SQL::Abstract
Retrieving columns

✦ programmer decides which
   - `columns => \@columns`  # arrayref
   - `columns => "col1, col2"` # string
   - `columns => "*"` # default

✦ no delayed fetching
✦ objects have variable size!
✦ runtime error if missing keys
  • for following joins
  • for updates and deletes
Associations

- declaration
- usage
Association declarations

MAD->Association(
    [qw/Author author 1 author_id/],
    [qw/Distrib distribs * author_id/]);

MAD->Association(
    [qw/Distrib distrib 1 distrib_id/],
    [qw/Module modules * distrib_id/]);
Associations

MAD->Association(
    [qw/Authorauthor 1 author_id/],
    [qw/Distribdistribs * author_id/]);

creates method
**Distrib::author** which returns a single object

creates method
**Author::distribs** which returns an arrayref

will generate **LEFT OUTER JOINS**
Role methods

```perl
foreach my $distrib (@$recent_distribs) {
    my $id     = $distrib->{distrib_id};
    my $author = $distrib->author();
    my $other_distribs = $author->distribs(
        -where => {distrib_id => {'<>' => $id}},
        -resultAs => 'iterator'
    );
}
```

same API as `TableOrView::select()`
Follow several roles at once

$rows = MAD
   ->ViewFromRoles(qw/Author distrib modules/)
   ->select(-where => ...);

$rows = $author
   ->selectFromRoles(qw/distribs modules/)
   ->(-where => ...);

new class created on the fly
aRow : AuthorDistribModule
Generated SQL

$rows = $author
    ->selectFromRoles(qw/distribs modules/)
    ->(-columns => [qw/distrib_name module_name/],
        -where => {d_release => {'<' => $date}});

SELECT distrib_name, module_name
FROM Distrib
    LEFT OUTER JOIN Module
    ON Distrib.distrib_id = Module.distrib_id
WHERE distrib.author_id = $author->{author_id}
AND d_release < $date
n-to-n Associations

MAD->Association(# from table1 to the link table
[qw/Distrib distrib 1 distrib_id/],
[qw/Dependency dependencies * distrib_id/]);

MAD->Association(# from table2 to the link table
[qw/Module module 1 module_id/],
[qw/Dependency dependencies * module_id/]);

MAD->Association(# n-to-n assoc with role names
[qw/Distrib distribs * dependencies distrib/],
[qw/Module modules * dependencies module/]);
Not covered here

- updates and transactions
- tree expansions and exports (XML, Json)
- column handlers for
  - data conversions (scalar or object)
  - data validation
- adding ad hoc methods
- criteria combinations: preselectWhere
- Views
- ...

See Also: DBIx::DataModel manual