

DBIx::Class

In the Real World

Nicholas Melnick

Frozen Perl 2010 / Minneapolis, Minnesota

Who Am I?

I am Nicholas Melnick.

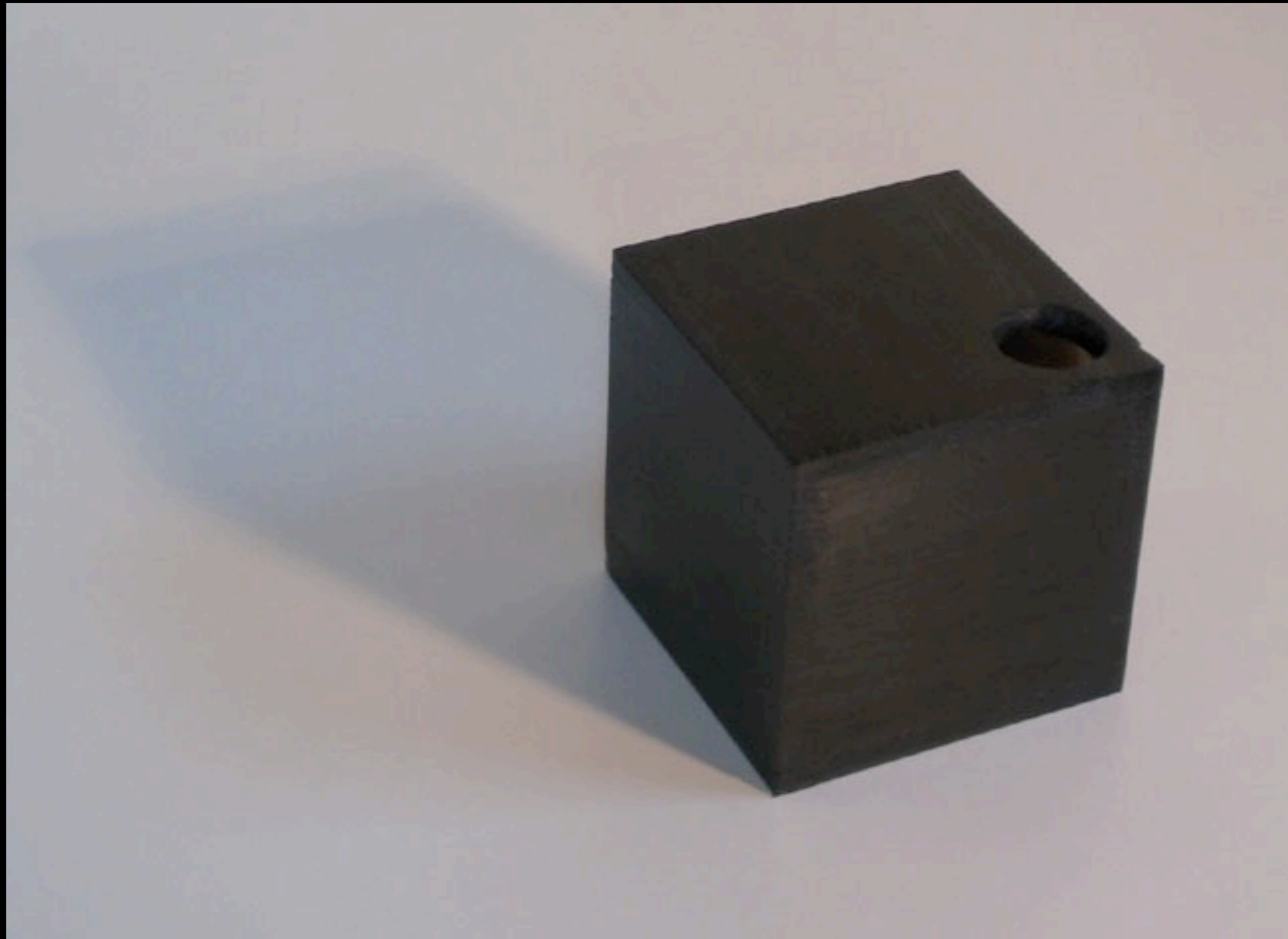
- Minneapolis.pm, formerly Seattle.pm/SPUG
- Own Sensical, Inc.
- Developer and architect of internet-based applications and nutty, ridiculous projects
- Guy

<http://www.abstractwankery.com> / @outzider

What is DBIx::Class?

- An Object Relational Mapper (ORM)
 - Turns relational data into objects
 - Makes data relationships sane
 - Makes your life easy
- Makes querying data more Perl-ish
 - Not too Perl-ish
- Brilliant

What is DBIx::Class?



So?

So?

What does it do for me?

The Quick Introduction:

The Quick Introduction:

Catalog Our Music.

Why music?

- Music has:
 - Many artists
 - Many albums
 - Many tracks on those albums
 - Many genres those tracks fit into

Those sound like relationships!

Write some SQL

```
CREATE TABLE artist (  
    artist_id integer primary key,  
    name        varchar(255)  
);
```

```
CREATE UNIQUE INDEX artist_name ON  
artist (name);
```

```
CREATE TABLE album (  
    album_id integer primary key,  
    artist_id integer,  
                REFERENCES artist  
    num_tracks integer,  
    name        varchar(255)  
);
```

```
CREATE UNIQUE INDEX album_name ON  
album (name);
```

```
CREATE TABLE genre (  
    genre_id integer primary key,  
    name        varchar(64)  
);
```

```
CREATE UNIQUE INDEX genre_name ON  
genre (name);
```

```
CREATE TABLE track (  
    track_id integer primary key,  
    album_id integer  
                REFERENCES album,  
    artist_id integer  
                REFERENCES artist,  
    genre_id integer  
                REFERENCES genre,  
    track_num integer,  
    year        integer,  
    date_added integer,  
    name        varchar(255),  
    UNIQUE (album_id, artist_id,  
            track_num)  
);
```

Make a Database

```
# sqlite3 catalog.db < catalog.sql
```

Make it Spicy

```
# sudo cpan DBIx::Class::Schema::Loader
```

Make it Spicy

```
# sudo cpan DBIx::Class::Schema::Loader
# dbicdump -o dump_directory=lib \
           -o debug=1 \
           -o use_namespaces=1 \
           Music::Schema \
           'dbi:SQLite:catalog.db'
```

```
# Create schema in lib/ directory
# Be verbose in the output
# When creating, put them in a 'Result' namespace
# Call it 'Music::Schema'
# Use this DBI connect string
```

Awesome.

```
lib/Music  
lib/Music/Schema  
lib/Music/Schema/Result  
lib/Music/Schema/Result/Album.pm  
lib/Music/Schema/Result/Artist.pm  
lib/Music/Schema/Result/Genre.pm  
lib/Music/Schema/Result/Track.pm  
lib/Music/Schema.pm
```

Really Awesome.

```
package Music::Schema::Result::Track;

use strict;
use warnings;

use base 'DBIx::Class::Core';

__PACKAGE__->table("track");
__PACKAGE__->add_columns(
    "track_id",
    { ... },
    "album_id",
    { ... },
    "artist_id",
    { ... },
    "genre_id",
    { ... },
    "track_num",
    { ... },
    "year",
    { ... },
    "date_added",
    { ... },
    "name",
    { ... },
);
__PACKAGE__->set_primary_key("track_id");
__PACKAGE__->belongs_to(
    "album",
    ...
);
__PACKAGE__->belongs_to(
    "artist",
    ...
);
__PACKAGE__->belongs_to(
    "genre",
    ...
);

1;
```

Use it.

```
#!/usr/bin/perl
```

```
use strict;  
use warnings;
```

```
use Music::Schema;
```

```
my $schema = Music::Schema->connect(  
    'dbi:SQLite:catalog.db'  
);
```

Access a table

```
my $artist_rs = $schema->resultset('Artist');
```

Add to table

```
$artist_rs->create();
```

Add Some Data

```
my $artist = $schema->resultset('Artist')->create({
    'name' => 'Nirvana',
});
my $album = $schema->resultset('Album')->create({
    'artist'      => $artist,
    'num_tracks' => 12,
    'name'        => 'Nevermind',
});
my $genre = $schema->resultset('Genre')->create({
    'name' => 'Alternative',
});
$album->tracks->create({
    'album'      => $album,
    'artist'     => $artist,
    'genre'      => $genre,
    'track_num' => 1,
    'year'       => 1991,
    'date_added' => time,
    'name'       => 'Smells Like Teen Spirit',
});
```

Save Some Time

```
my $album_two = $schema->resultset('Album')->create({
    'artist' => {
        'name' => 'Stone Temple Pilots',
    },
    'num_tracks' => 11,
    'name' => 'Purple',
});
$album_two->tracks->create({
    'album' => $album_two,
    'artist' => $album_two->artist,
    'genre' => {
        'name' => 'Alternative',
    },
    'track_num' => 1,
    'year' => 1994,
    'date_added' => time,
    'name' => 'Meatplow',
});
```

Data!

```
SELECT * FROM artist;
```

```
artist_id | name
```

```
-----+-----  
1         | Nirvana  
2         | Stone Temple Pilots
```

```
SELECT * FROM album;
```

```
album_id | artist_id | num_tracks | name
```

```
-----+-----+-----+-----  
1        | 1         | 12         | Nevermind  
2        | 2         | 11         | Purple
```

```
SELECT * FROM genre;
```

```
genre_id | name
```

```
-----+-----  
1        | Alternative
```

```
SELECT * FROM track;
```

```
track_id | album_id | artist_id | genre_id | track_num | year | date_added | name
```

```
-----+-----+-----+-----+-----+-----+-----+-----  
1        | 1        | 1         | 1         | 1         | 1991 | 1262843965 | Smells Like Teen S...  
2        | 1        | 1         | 1         | 2         | 1991 | 1262843965 | In Bloom  
3        | 1        | 1         | 1         | 3         | 1991 | 1262843965 | Come As You Are  
4        | 1        | 1         | 1         | 4         | 1991 | 1262843965 | Breed  
5        | 2        | 2         | 1         | 1         | 1994 | 1262843965 | Meatprow
```

Get Data Out

- search

```
my $set = $schema->resultset('track')->search(  
    {  
        [ what you're looking for ]  
    },  
    {  
        [ filter, link, relation, and slim ]  
    }  
);
```

- retrieve

```
my $row = $set->first();    # Result 1  
$row    = $set->next();    # Result 2
```

Get Data Out

- Get all tracks by artist

```
my $set = $schema->resultset('track')->search(
    {
        'artist.name' => 'Nirvana'
    },
    {
        'join' => 'artist'
    }
);
while ( my $track = $set->next() ) {
    printf( "Track %d: %s, from the album %s\n",
           $track->track_num, $track->name,
           $track->album->name );
}
```

Get Data Out

- Get all tracks by artist

```
my $artist = $schema->resultset('artist')->search({
    'name' => 'Nirvana'
});
my $set = $artist->search_related('tracks');
while ( my $track = $set->next() ) {
    printf( "Track %d: %s, from the album %s\n",
           $track->track_num, $track->name,
           $track->album->name );
}
```

Success!

Track 1: Smells Like Teen Spirit, from the album Nevermind

Track 2: In Bloom, from the album Nevermind

Track 3: Come As You Are, from the album Nevermind

Track 4: Breed, from the album Nevermind

What was \$set?

```
my $set = $schema->resultset('track')->search(
```

- ResultSets are the query definition of a collection of records
 - This could be a table
 - This could be a view
 - This could be the result of your search

ResultSets

- ResultSets can be modified with:
 - ->search
 - ->delete
 - ->update
 - ->count
 - ->create

ResultSets

- ResultSets can be chained:
 - `$rs1 = $schema->resultset('Artist');`
 - `$rs2 = $rs1->search({
 'name' => { 'LIKE' => 'Nir%' }
});`
 - `$rs3 = $rs2->search({
 'artist_id' => { '>' => 3 }
})->count();`
That's zero, by the way.

ResultSets

- ResultSets can help you create:
 - `$rs1 = $schema->resultset('Album');`
 - `$rs1 = $rs1->search({
 'artist' => {'name' => 'Nirvana' },
});`
 - `$album = $rs1->create({
 'name' => 'Bleach'
});`

ResultSets

- ResultSets can be filtered and paged:
 - `$rs1 = $schema->resultset('Track');`
 - `$rs1 = $rs1->search(undef,
 { 'rows' => 30 });`
 - `$rs2 = $rs1->page(1);`
 - `$rs3 = $rs1->page(2);`
- `$row = $rs3->first;`

Relationships

- belongs_to
- has_one
- has_many
- might_have
- many_to_many

Relationships

- Relationships are ResultSets
- `my $song_count = $track->artist->albums->first->tracks->count();`

Result and ResultSet

```
$stats{'usersTotal'} = $schema->resultset('User')->search(undef)->count;
$stats{'usersConfirmed'} = $schema->resultset('User')->search({'user_status_id' => 5})->count;
$stats{'usersToTransition'} = $schema->resultset('User')->search({'user_status_id' => 7})->count;
$stats{'usersEmailed'} = $schema->resultset('User')->search({'user_status_id' => 10})->count;
$stats{'usersNewPassword'} = $schema->resultset('User')->search({'user_status_id' => 6})->count;
$stats{'usersWithMail'} = $schema->resultset('User')->search({'has_mail' => 1})->count;
$stats{'confirmedToday'} = $schema->resultset('User')->search({'date_confirmed' => { '>=', $date }})->count;
$stats{'signupsToday'} = $schema->resultset('User')->search({'date_created' => { '>=', $date }})->count;
$stats{'userLoginsToday'} = $schema->resultset('User')->search({'date_last_on' => { '>=', $date }})->count;
$stats{'usersActiveToday'} = $schema->resultset('Session')->search({'last_access' => { '>=', $date }})->count;
```

```
$stats{'profiles'} = $schema->resultset('Profile')->search(undef)->count;
```

```
$stats{'boardMessages'} = $schema->resultset('Boardpost')->search(undef)->count;
$stats{'boardMessagesToday'} = $schema->resultset('Boardpost')->search({'datecreated' => { '>=', $date }})->count;
```

```
$stats{'emails'} = $mail_schema->resultset('MailHeader')->search(undef)->count();
$stats{'emailsToday'} = $mail_schema->resultset('MailHeader')->search({'date_indexed' => { '>=', $date }})->count;
```

```
$stats{'statistics'} = \%info;
$stats{'usersActive'} = $schema->resultset('Session')->search({'last_access' => { '>=', strftime('%Y-%m-%d %H:%M:%S', localtime(time - (60 * 20))) }})->count();
$stats{'usersFalling'} = $schema->resultset('Session')->search({'last_access' => { '>=', strftime('%Y-%m-%d %H:%M:%S', localtime(time - (60 * 30))) }})->count();
$stats{'usersFalling'} -= $stats{'usersActive'};
$stats{'usersIdle'} = $schema->resultset('Session')->search({'last_access' => { '<', strftime('%Y-%m-%d %H:%M:%S', localtime(time - (60 * 30))) }})->count();
```

Result and ResultSet

- Classes that you can extend
- Make your life easier
- **Make your code cleaner**

Result

- Represents a table AND a record
- Can create triggers and accessors
- Can alter the behavior of fields

Result: Inflate

- Translate data to something more useful
 - Integers to IP addresses and back
 - Serialization of data structures
 - Dates to DateTime objects

Inflate: DateTime

```
package Music::Schema::Result::Track;
```

```
use strict;
```

```
use warnings;
```

```
use base 'DBIx::Class::Core';
```

```
__PACKAGE__->table("track");
```

```
__PACKAGE__->add_columns(
```

```
    "track_id",
```

```
    { ... },
```

```
    "album_id",
```

```
    { ... },
```

```
    "artist_id",
```

```
    { ... },
```

```
    "genre_id",
```

```
    { ... },
```

```
    "track_num",
```

```
    { ... },
```

```
    "year",
```

```
    { ... },
```

```
    "date_added",
```

```
    { ... },
```

```
    "name",
```

```
    { ... },
```

```
);
```

Inflate: DateTime

```
package Music::Schema::Result::Track;

use strict;
use warnings;

use base 'DBIx::Class::Core';

__PACKAGE__->table("track");
__PACKAGE__->load_components(qw/DateTime::Epoch/);
__PACKAGE__->add_columns(
    "track_id",
    { ... },
    "album_id",
    { ... },
    "artist_id",
    { ... },
    "genre_id",
    { ... },
    "track_num",
    { ... },
    "year",
    { ... },
    "date_added",
    { inflate_datetime => 1, },
    "name",
    { ... },
);
```

Inflate: DateTime

```
my $set = $schema->resultset('Track')->search();
while ( my $track = $set->next() ) {
    printf( "Track %d was added on %s at %s.\n",
           $track->track_num, $track->date_added->ymd,
           $track->date_added->hms );
}
```

Inflate: DateTime

```
my $set = $schema->resultset('Track')->search();
while ( my $track = $set->next() ) {
    printf( "Track %d was added on %s at %s.\n",
           $track->track_num, $track->date_added->ymd,
           $track->date_added->hms );
}
```

#####

```
Track 1 was added on 2010-01-07 at 07:55:47.
Track 2 was added on 2010-01-07 at 07:55:47.
Track 3 was added on 2010-01-07 at 07:55:47.
Track 4 was added on 2010-01-07 at 07:55:47.
Track 1 was added on 2010-01-07 at 07:55:47.
```

Inflate: DateTime

```
my $set = $schema->resultset('Track')->search();
while ( my $track = $set->next() ) {
    printf( "Track %d was added on %s at %s.\n",
           $track->track_num, $track->date_added->ymd,
           $track->date_added->hms );
    $track->set_day(25);
    $track->update();
}
```

Inflate Your Own

```
package Music::Schema::Result::Track;

...

__PACKAGE__->inflate_column('column_name', {
    'inflate' => sub { return 'inflated' },
    'deflate' => sub { return 'deflated' },
});
```

Model-Level Triggers

- Updating one field updates another:

```
sub store_column {  
  my ( $self, $column_name, $value ) = @_;  
  $self->date_updated( \ 'now()' );  
  $self->next::method( $name, $value );  
}
```

- Pre-fill fields on create

```
sub new {  
  my ( $class, $attrs ) = @_;  
  $attrs->{'date_created'} = \ 'now()'  
  unless ( defined $attrs->{'date_created'} );  
  return $class->next::method( $attrs );  
}
```

Arbitrary Accessors

- “Virtual Columns”
- Access functions within your record

```
sub pretty {  
    my ( $self ) = @_;  
    return sprintf( “%s”, by %s’, $self->name, $self->artist->name );  
}
```

- Access anywhere in your database

```
sub random_album_name {  
    my ( $self ) = @_;  
    return $self->result_source->schema->resultset('Album')->search( undef, { 'order_by' => \'RANDOM()\' } )->first->name;  
}
```

Objectify your SQL

- Complex queries are sometimes hard
- DBIx::Class can make them more hard
- Adding SQL to your code is ugly

Objectify your SQL

```
package Music::Schema::Result::iTunes;
use strict;
use warnings;
use base qw/DBIx::Class::Core/;

__PACKAGE__->table_class('DBIx::Class::ResultSource::View');
__PACKAGE__->table('dummy');
__PACKAGE__->add_columns(qw/ track_id name artist album track_num genre /);
__PACKAGE__->result_source_instance->is_virtual(1);
__PACKAGE__->result_source_instance->view_definition(q{
    SELECT      track_id, track.name, artist.name AS artist,
               album.name AS album, track_num, genre.name AS genre
    FROM        track
    LEFT JOIN   artist ON track.artist_id = artist.artist_id
    LEFT JOIN   album ON track.album_id = album.album_id
    LEFT JOIN   genre ON track.genre_id = genre.genre_id
});

1;
```

Objectify your SQL

```
my $out = "%-3s | %-25s | %-19s | %-15s | %-5s | %-16s\n";
my $set = $schema->resultset('iTunes')->search();
printf( $out, 'ID', 'Title', 'Artist', 'Album', 'Track', 'Genre' );
print '-' x 93 . "\n";
while ( my $track = $set->next() ) {
    printf( $out,
            $track->track_id,
            $track->name,
            $track->artist,
            $track->album,
            $track->track_num,
            $track->genre,
    );
}
```

Objectify your SQL

```
my $out = "%-3s | %-25s | %-19s | %-15s | %-5s | %-16s\n";
my $set = $schema->resultset('iTunes')->search();
printf( $out, 'ID', 'Title', 'Artist', 'Album', 'Track', 'Genre' );
print '-' x 93 . "\n";
while ( my $track = $set->next() ) {
    printf( $out,
           $track->track_id,
           $track->name,
           $track->artist,
           $track->album,
           $track->track_num,
           $track->genre,
    );
}
```

ID	Title	Artist	Album	Track	Genre
1	Smells Like Teen Spirit	Nirvana	Nevermind	1	Alternative
2	In Bloom	Nirvana	Nevermind	2	Alternative
3	Come As You Are	Nirvana	Nevermind	3	Alternative
4	Breed	Nirvana	Nevermind	4	Alternative
5	Meatplow	Stone Temple Pilots	Purple	1	Alternative

ResultSet

- Same thing
- Create convenience methods
- Clean up your freakin' codebase

ResultSet

```
package Music::Schema::ResultSet::Track;  
  
use strict;  
use warnings;  
use base 'DBIx::Class::ResultSet';  
  
1;
```

ResultSet: Convenience

```
package Music::Schema::ResultSet::Track;

use strict;
use warnings;
use base 'DBIx::Class::ResultSet';

sub starts_with {
    my ( $self, $starts_with ) = @_;

    return $self->search({
        'name' => { 'LIKE' => $starts_with . '%' },
    });
}

1;
```

ResultSet: Convenience

```
my $set = $schema->resultset('Track')->search()->starts_with('B');
while ( my $track = $set->next() ) {
    printf( "Track %d: %s, from the album %s\n",
           $track->track_num, $track->name,
           $track->album->name );
}
```

####

Track 4: Breed, from the album Nevermind

Now what?

Time to Scale

- Many times, SQLite isn't enough
- Moving to a new database engine sucks
- You'll probably need to do more than this

Bam, PostgreSQL

```
#!/usr/bin/perl

use strict;
use warnings;
use Music::Schema;

my $schema = Music::Schema->connect(
    'dbi:Pg:dbname=catalog',
    'postgres',
    'postgres',
);
$schema->deploy();

1;
```

Bam, PostgreSQL

```
# createdb catalog  
# perl ./deploy.pl
```

```
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index  
"artist_pkey" for table "artist"
```

```
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "genre_pkey"  
for table "genre"
```

```
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "album_pkey"  
for table "album"
```

```
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "track_pkey"  
for table "track"
```

```
NOTICE: CREATE TABLE / UNIQUE will create implicit index  
"album_id_artist_id_track_num_unique" for table "track"
```

Go Forth.

Go Forth.

(not FORTH)

Go Forth.

(not FORTH)

(thank you very much.)