



ANGERIN DATABASE GODS

MAGGIE NELSON
MAY 23, 2008

Scope

relational databases

such as:

MySQL

Oracle

PostgreSQL

OOP-based PHP frameworks

DDL

Data Definition Language

create

alter

drop

DML

Data Modification Language

insert

update

delete

SQL

Structured Query Language

select

CRUD!

C

create (insert)

DML

R

read (select)

SQL

U

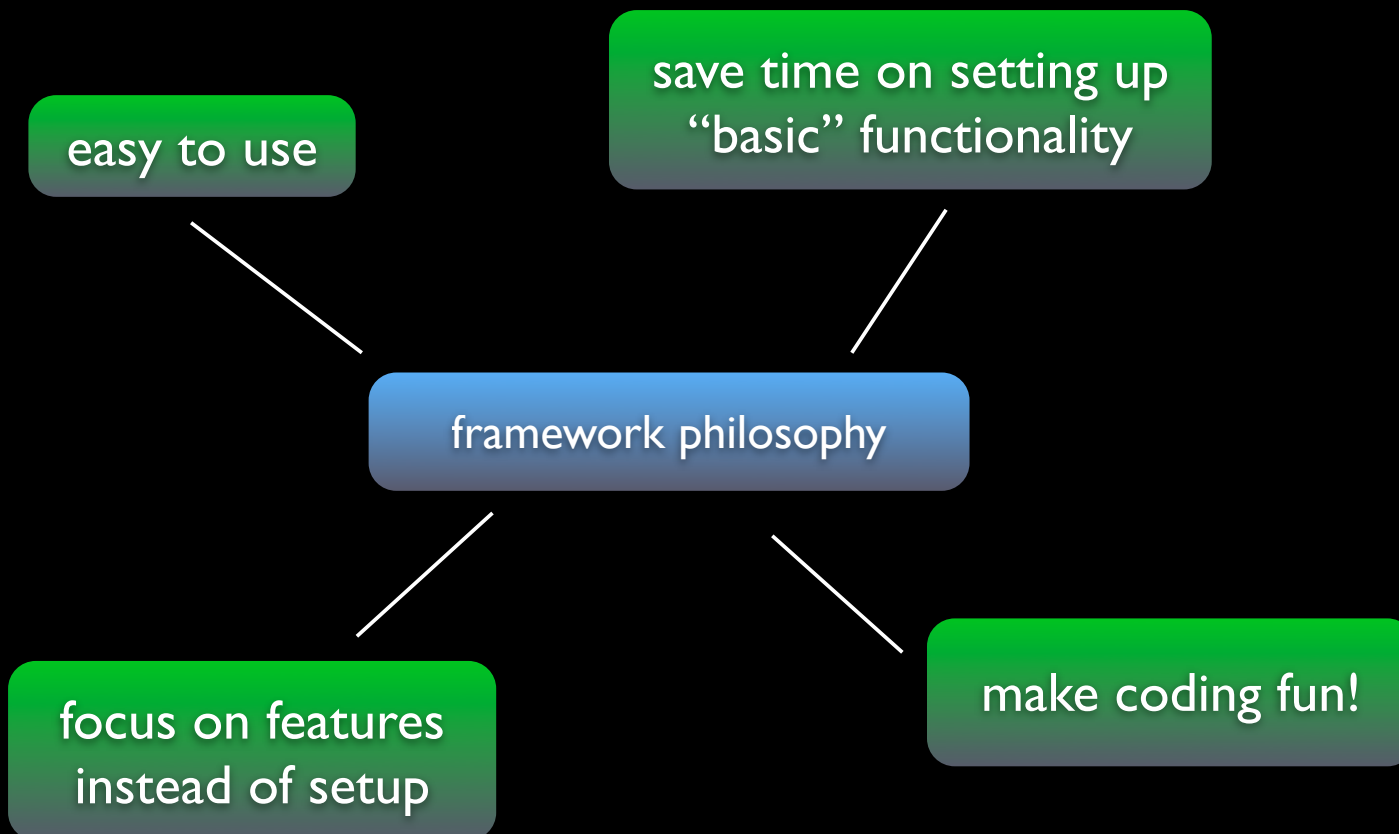
update

DML

D

delete

DML



easy to turn prototypes
into full-fledged apps

developers may pay less
attention to how the
entire system works

framework dangers

automagical functionality may
make debugging difficult

understand what your
framework is doing

pay attention to all parts
of your system

frameworks are a good tool

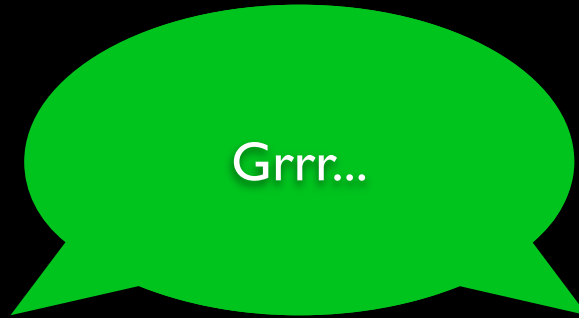
easy now != easy forever

good software design
practices still apply

relations vs. objects



database



Grrr...



OOP



database

scalar values

normalized data

concepts (objects)
stored over many
tables

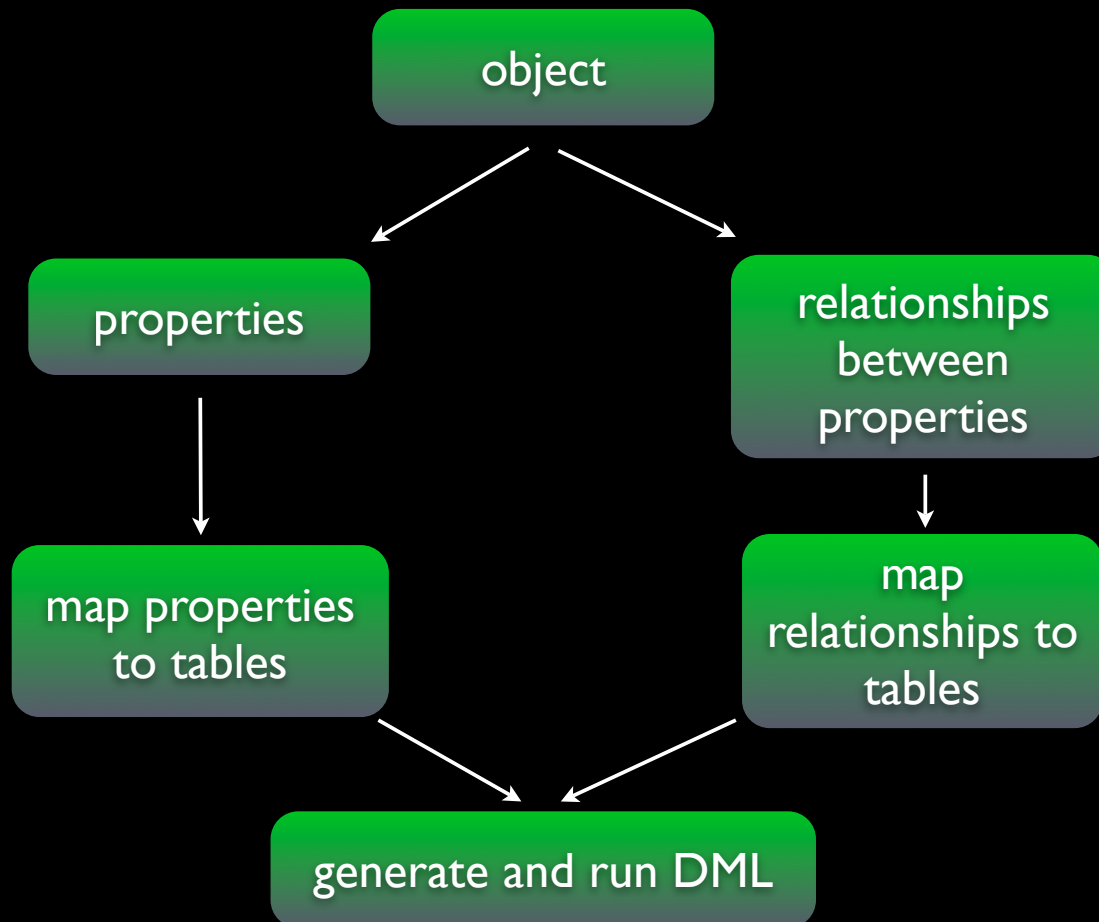
OOP

one object stores
many values

an object defines
behavior

ORM: object-relational mapping

translate objects to relations (tables)



ORM: object-relational mapping

translate relations (tables) to objects

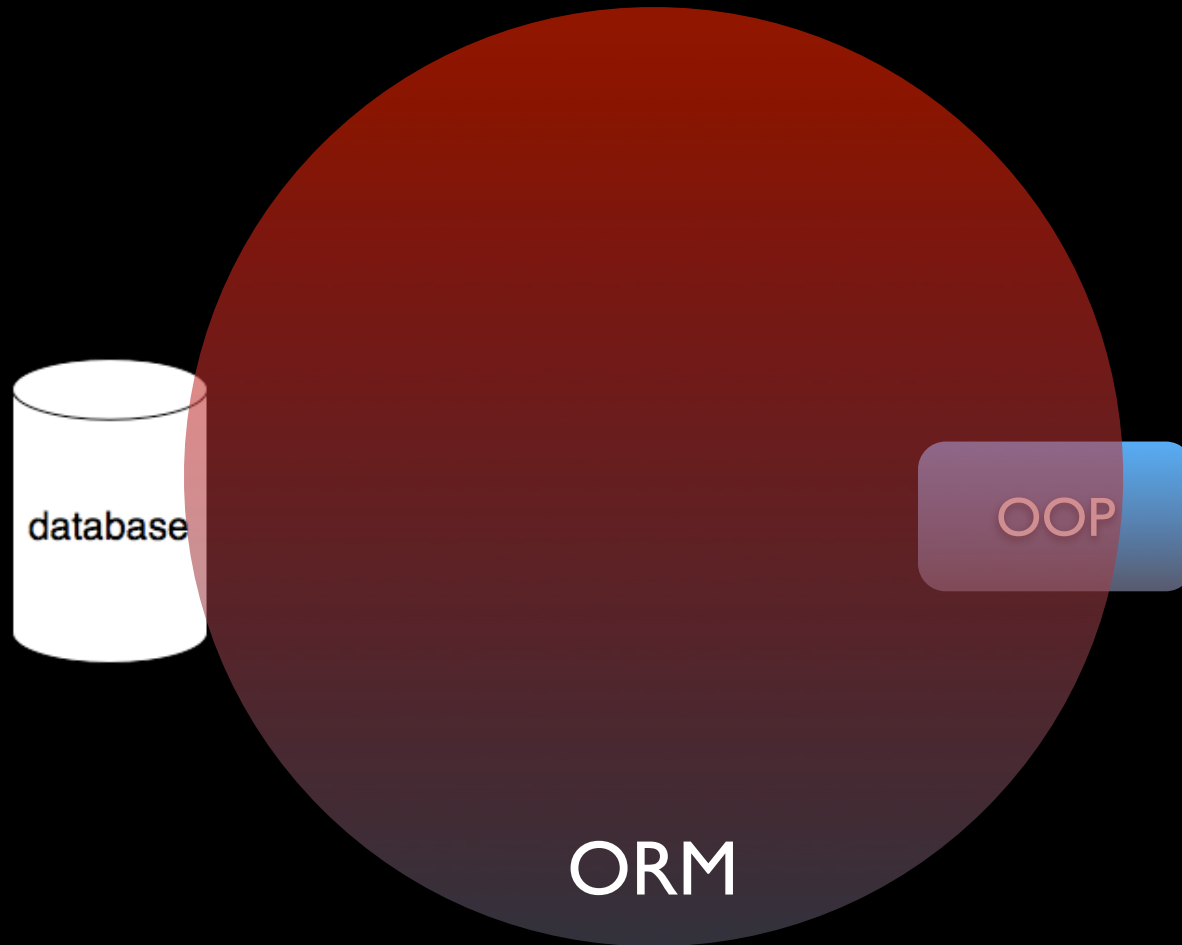
tables

figure out which tables store data for the object and its relationships

generate and run SQL

translate results into objects

db-app relationship is only as good as the ORM



a word about normalization

normalization



minimization

of

duplication

of

information

not NF

books

title	author
“Cryptonomicon”, “Snow Crash”	Neal Stephenson
“The Name of the Rose”	Umberto Eco
“The Lion, the Witch and the Wardrobe”	C.S. Lewis

1NF: free of repeating groups

books

title	author
"Cryptonomicon"	Neal Stephenson
"Snow Crash"	Neil Stephenson
"The Name of the Rose"	Umberto Eco
"The Lion, the Witch and the Wardrobe"	C.S. Lewis

1NF -> 2NF

books

title	author	author birthdate
"Cryptonomicon"	Neal Stephenson	1959
"Snow Crash"	Neil Stephenson	1959
"The Name of the Rose"	Umberto Eco	1932
"The Lion, the Witch and the Wardrobe"	C.S. Lewis	1898

2NF: depend on entire PK

books

title	author_id
"Cryptonomicon"	1
"Snow Crash"	1
"The Name of the Rose"	2
"The Lion, the Witch and the Wardrobe"	3

authors

id	name	birthdate
1	Neal Stephenson	1959
2	Umberto Eco	1932
3	C.S. Lewis	1898

3NF: data about the key

books

title	author_id
"Cryptonomicon"	1
"Snow Crash"	1
"The Name of the Rose"	2
"The Lion, the Witch and the Wardrobe"	3

authors

id	name	birthdate
1	Neal Stephenson	1959
2	Umberto Eco	1932
3	C.S. Lewis	1898

while designing your database:

2 steps to figure out how much to normalize:

1

normalize

2

denormalize

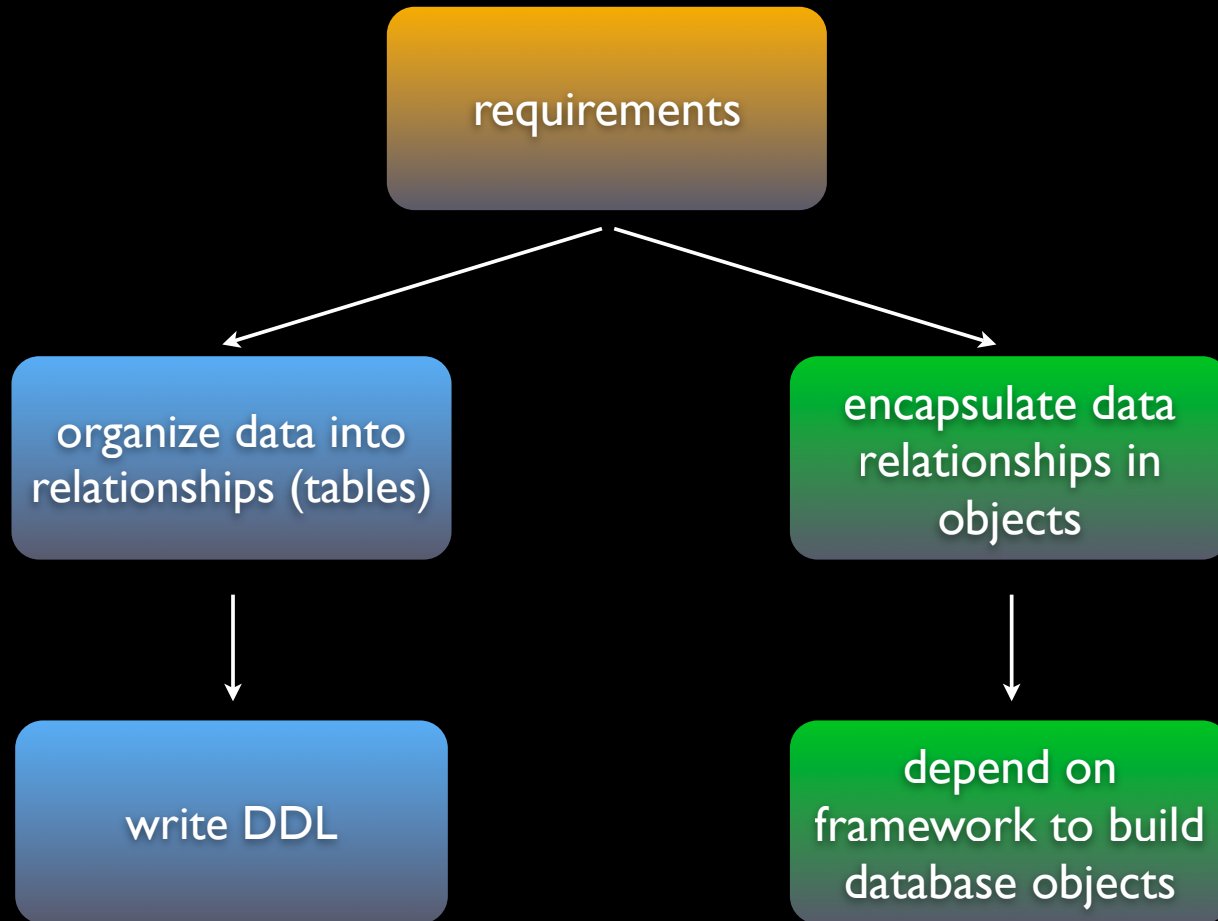
most web applications fall
between 1NF and 3NF

DDL (Data Definition Language)

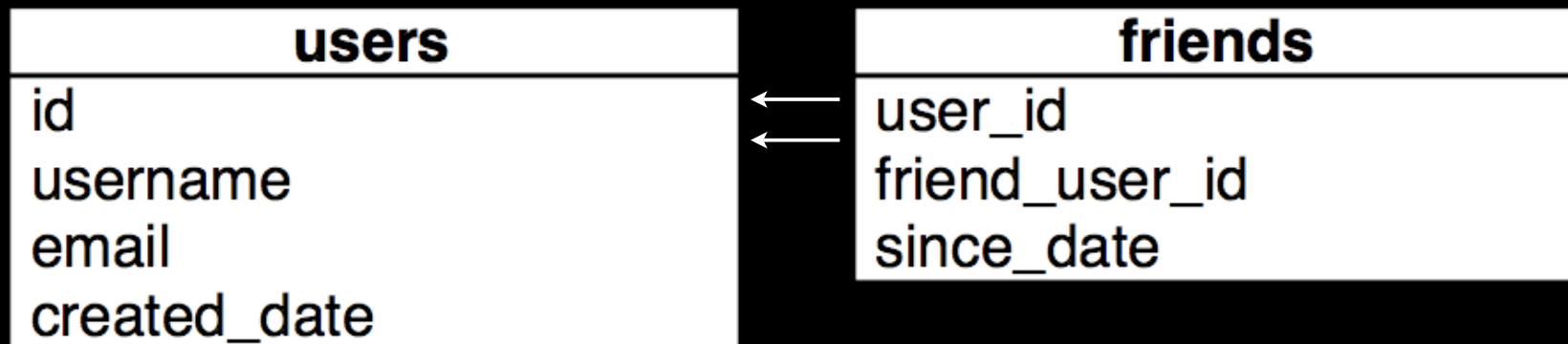
CREATE

ALTER

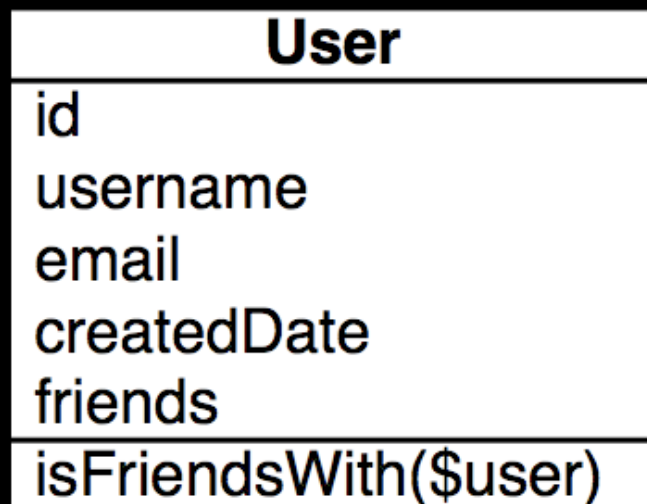
creating database objects



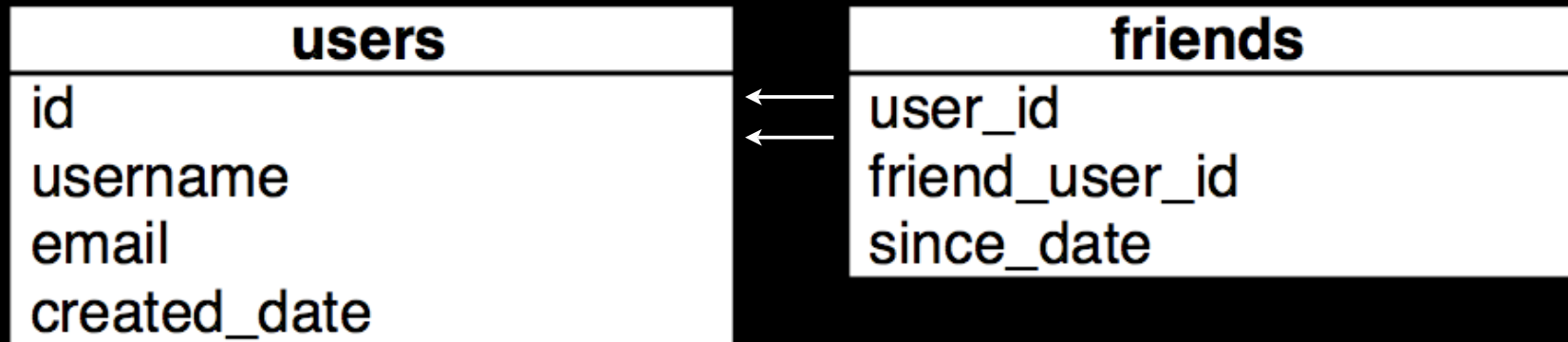
tables



objects



tables



clear foreign key relationships

easy to notice need for
additional constraints

normalized data

able to limit retrieval of friends

objects

User
id
username
email
createdDate
friends
isFriendsWith(\$user)

easy to use

must retrieve all friends

allows for duplication of data
(denormalization)

create table as select

tablespace

partitioning

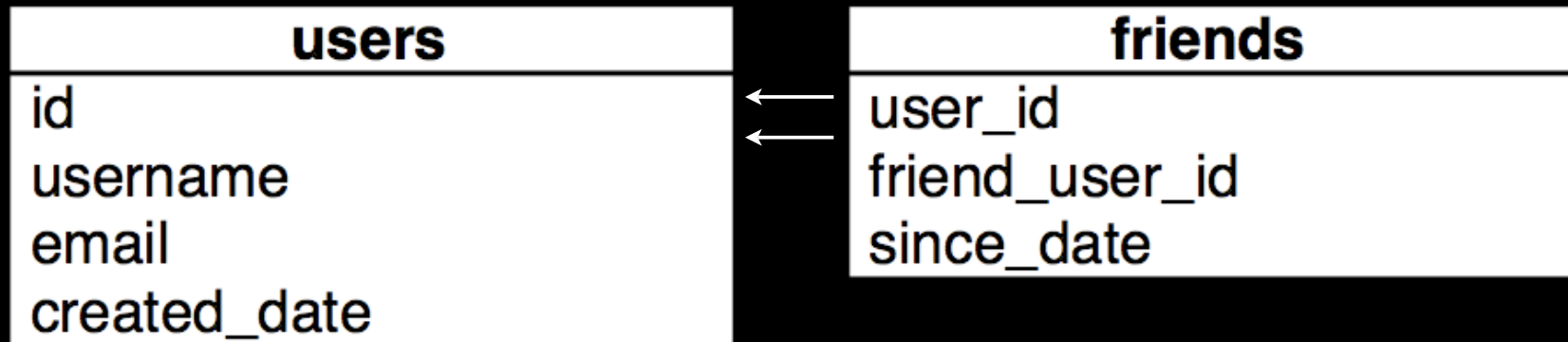
Frameworks don't account for ALL possible options when creating database objects

cluster

organization

privileges

referential integrity



primary keys

- `users_id_pk`

not null constraints

- on all fields here

foreign keys

- `friends_user_id_fk`
- `friends_friend_user_id_fk`

unique constraints

- `friends_uk (user_id, friend_user_id)`

Frameworks might not create all required constraints.

It's not easy to name your constraints.

good database design

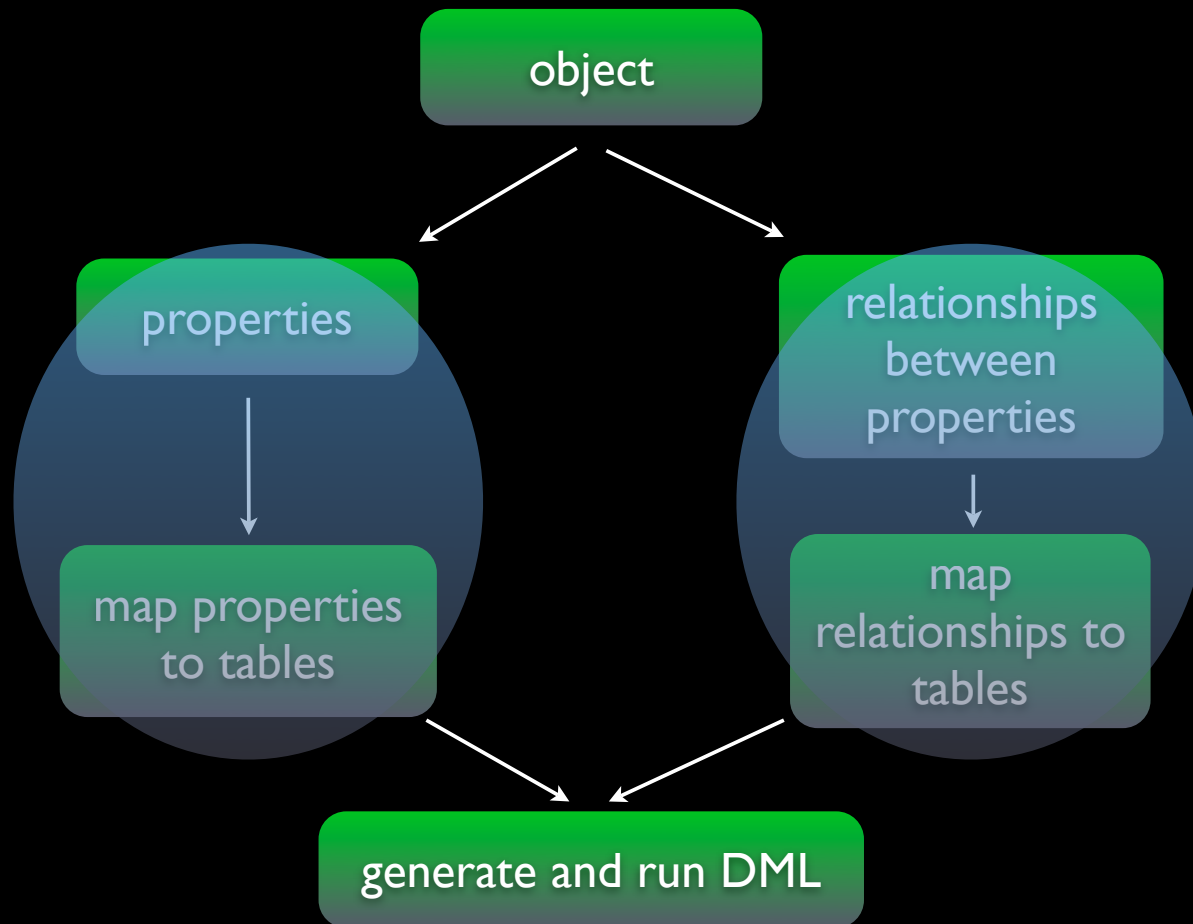


opportunities for optimization

“Premature optimization is the root of all evil.”
- D. Knuth

ORM: object-relational mapping

translate objects to relations (tables)



Pattern: Active Record

an object that encapsulates a row
from a database table or view

DML (Data Modification Language)

INSERT

UPDATE

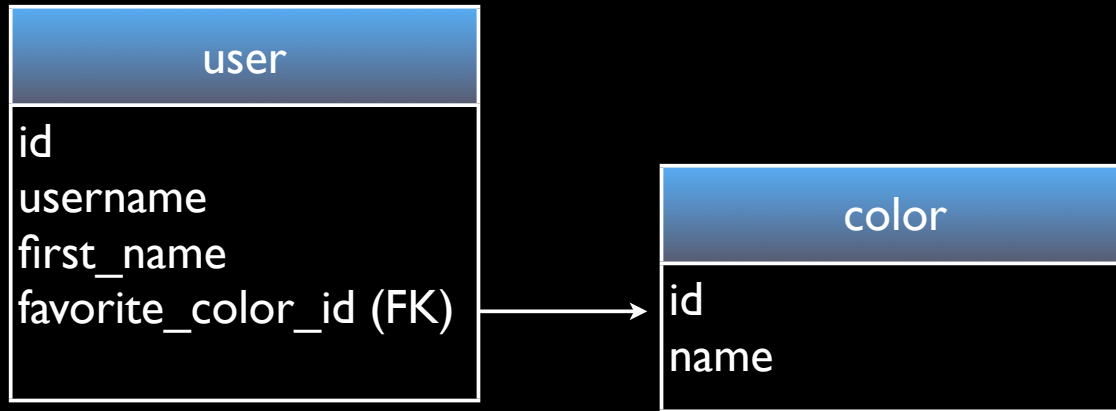
DELETE

user
id
username
firstName
favoriteColor
save()

```
CREATE TABLE user (  
  id int,  
  username varchar(255),  
  firstName varchar(255),  
  favoriteColor varchar(255)  
);
```

user
id
username
firstName
favoriteColor
save()

```
$user->favoriteColor = 'blue';  
$user->save();
```



```
UPDATE user
  SET favorite_color_id = :color_id
WHERE id = :user_id;
```

:color_id??

user
id
username
firstName
favoriteColor
save()

```
foreach($usersWhoLikeBlue as $user) {  
    $user->favoriteColor = 'red';  
    $user->save();  
}
```

user
id
username
(20 fields)
save()

```
SELECT *  
FROM user  
WHERE favorite_color = 'blue';
```

```
print "These people like blue!";  
foreach($usersWhoLikeBlue as $user) {  
    // 20 fields available per $user  
    print($user->username);  
}  
print "bye!";
```

SQL (Structured Query Language)

SELECT

Pattern: Table Data Gateway

without:

developers write
SQL / DML

developer executes
SQL / DML

return results

with:

DBAs write SQL /
DML for table

SQL / DML is encapsulated in
TDG (1 instance of TDG
handles all rows in the table)

developers use TDG
class instead of accessing
tables directly

return results

Pattern: Table Data Gateway

without:

developers write
SQL / DML

developer executes
SQL / DML

return results

with:

DBAs write SQL /
DML for table

SQL / DML is encapsulated in
TDG (1 instance of TDG
handles all rows in the table)

developers use TDG
class instead of accessing
tables directly

return results

In theory, TDG is great, but you might end up getting a lot more data than you need.

Generating SQL

what does this mean?

```
$select = $db->select()  
  ->from(array('p' => 'products'),  
        array('product_id'))  
  ->join(array('l' => 'line_items'),  
        'p.product_id = l.product_id',  
        array('line_items_per_product' => 'COUNT(*)'))  
  ->group('p.product_id');
```

this may be clearer...

```
SELECT p."product_id", COUNT(*) AS line_items_per_product
FROM "products" AS p JOIN "line_items" AS l
  ON p.product_id = l.product_id
GROUP BY p.product_id;
```

but this is nicer

```
select p.product_id,  
       count(*) line_items_per_product  
  from products p,  
       line_ites l  
 where p.product_id = l.product_id  
 group by p.product_id;
```

compare...

```
$select = $db->select()  
  ->from(array('p' => 'products'),  
        array('product_id'))  
  ->join(array('l' => 'line_items'),  
        'p.product_id = l.product_id',  
        array('line_items_per_product' => 'COUNT(*)'))  
  ->group('p.product_id');
```

```
select p.product_id,  
       count(*) line_items_per_product  
  from products p,  
       line_ites l  
 where p.product_id = l.product_id  
 group by p.product_id;
```

and how about non-standard SQL?

standard SQL

```
SELECT u.first_name,  
       u.last_name  
FROM user u  
LEFT OUTER JOIN occupation o USING (u.occupation_id = o.id)  
WHERE u.last_name = 'Eco'
```

Oracle outer join syntax

```
SELECT u.first_name,  
       u.last_name,  
       FROM user u,  
       occupation o  
WHERE o.id (+) = u.occupation_id  
      AND u.last_name = 'Eco'
```

and a variety of RDBMS-specific functions

e.g. Oracle's analytic functions

Most frameworks give in and provide a way to write straight-up SQL anyway

optimization

Not really compatible with the idea of query tuning.

denormalization of data

frameworks might not understand or support views

views may contain data to populate more than one ActiveRecord

mapping denormalized data to full objects can be tough

limiting database access

ORM may require a lot of database round trips

ORM can easily ignore what DBAs and database developers achieved in providing a scalable, high-performing system

Model View Controller

View

- easy: just display data

Model

- is it like ActiveRecord?
- is it like a collection of records?
- does it talk to the DB?

Controller

- do actions talk to the DB?
- does it retrieve models?
- does it create models?

Can you do it all in one query?

Remember: best tool for the job

understand what your
framework is doing

pay attention to all parts
of your system

frameworks are a good tool

easy now != easy forever

good software design
practices still apply

Questions?

Thanks!

more info:
maggienelson.com