ORMs and Their ~~Discotheques~~ Discontents

Christophe Pettus

PGXPUG Day OSCON 2010

July 18, 2010

Has this ever happened to you?

- "This query is running way too slowly. God, RDBMSes suck!"
- *"Well, you just need to change the WHERE clause…"*
- "I can't change the SQL. We're using an…"


ORM

ORMs and Their Discontents

- Why ORMs at all?
- What are the problems?
- What can we do to keep from killing app developers?

Object-Relational Mapper

- Hibernate (Java)
- ActiveRecord (Ruby, doh)
- Django ORM, SQLAlchemy (Python)
- Propel (PHP)
- Core Data (Objective-C)
- Taligent Object Storage Framework (C++, three years of my life down the drain)

What is OO programming?

- Not inheritance.
- Not “messaging.”
- Not typing.

Encapsulation.

- Object-Oriented Programming is fundamentally about encapsulation.
- The basic building block is an object that exposes operations.
- The object defines behavior, and you bring the data.

The Relational Model

- The basic unit is a tuple, organized into tables.
- The data of the tuple is fully exposed.
- The model relies on the ability to "pull apart" tuples.
- The database holds data, and you bring the behavior.

Pity the Poor ORM

- An Object-Relational Mapper has to bridge these worlds.
  - OO class maps to a DB table.
  - OO data member maps to DB column.
  - And everything else is kind of random.

Object Structures are Graphs.

- OO is all about in-memory objects with references (pointers, what have you) to other objects.
- The set of references is relatively static to a particular class.

The Relational Model Isn't.

- Tables come and go all the time.
- The "tables" as such are really just a privileged set of relations that have really long lifetimes.
- Foreign keys are more about data integrity, less about permanent relationships.

The Application Programmer's Lament.

I Just Want This Object to Be Stored On Disk.

Is that too much to ask?

All ORMs Started That Way.

- The application was written in an OO language.
- They had an RDBMS.
- They needed to store objects.
- Hammer implies nail.

App Programmers Aren't Stupid.

- Well, no more than most people.
- But they are lazy.
- But then again, who isn't?

Let's Face It, Writing Code Like This Sucks.

```
cursor* curs;
curs = db_connection->create_cursor();

customer_order *order = new(customer_order);

if (curs.execute("SELECT * from customer_order WHERE order_id=123")) {
    result_set* results;
    results = curs->fetch_results();

    customer_order->order_id = results->fetch_column("order_id");
    customer_order->customer_id = results->fetch_column("customer_id");
    customer_order->date_placed = results->fetch_column("date_placed");
    // ??? Need to finish.  First programmer quit to become
    // ??? a tour guide in Slovakia.
}
```

Who Wouldn't Rather Write This?

```
customer_order* order = customer_order.retrieve(123);
order->cancel();
   // Didn't want that loser's business anyway.
order->save();
   // Off for a latte!
```

Non-Tech Reasons.

- ORMs promise database independence.
  - Mephistopheles gave Faust a great spec sheet, too.
- App programmers hate SQL. They really, really, really `hate` SQL.
  - SQL is taught as a command language, not a discipline.

What could possibly go wrong?

- "Great! Oh, we need to increment the order_age field each night at midnight."
- "Could we get a report of all open orders where the customer has prepaid more than 50% of the total?"
- "Why is the database running so slowly? God, RDBMSes suck! Let's use CouchDB."

The More Rows, the Bigger the Problem.

- ORMs generally break down on highly relational or large multi-row operations.
- The naïve approach is almost always wrong.
  - ORMs encourage pathological iteration.
- Various ORMs have grown various tools to deal with this…
  - But at that point, they're just weird-syntax SQL.

Transaction (Mis-) Management

- ORMs generally have bizarre transaction models.
- “Each operation its own transaction” seems to be a typical default.
- Transaction management tools are often made to seem like a black art.

```
address = Address(street_address="1112 E Broad St",
city="Westfield", state="NJ", zip="07090")

address.save()

order = Order(customer_name="Gomez Addams",
shipping_address=address)

order.save()
```

```
BEGIN;

INSERT INTO Address VALUES (...);

COMMIT;
```

```
BEGIN;

INSERT INTO Order VALUES (...);

COMMIT;
```

The Full Flexibility of SQL… Except That.

- ORMs frequently impose restrictions on the database schema.
- Example: Django ORM doesn't allow composite primary keys on rows.
- Triggers? Constraints? Stored Procedures?

I'm Helping!

- Database-agnosticism frequently means the ORM tries too hard.
- Example: Django's ORM does cascaded deletion across foreign keys…
- … even if the underlying DBMS supports it.

Bypass Surgery.

- Just about every ORM allows direct access to the SQL layer.
- Of course, ORMs are also doing in-memory caching, and you're on your own for cache invalidation.
- "The great thing about this model of car? You can still walk!"

The Slice Non-Problem.

- The “SELECT *” problem is not a core problem.
- But all ORMs try to fix it anyway.
  - Breaks encapsulation.
  - Probably reveals a bad schema design.

So, Why Should We Care?

- DB administrators and architects are routinely called in to fix ORM-related problems.
- Problem with ORMs (or their use) are attributed to SQL, not the ORM.
- ORM-think is one of the driving forces behind "NoSQL" databases.

What To Do?

- ORMs are not going away — nor should they.
- "Better ORMs" are not the answer.
  - ORMs have been around since the early 1990s.
  - If we could fix it that way, we would have by now.

It's an Education Problem.

- SQL is treated as a command language on a par with `bash`.
- Web developers tend to be focused on the front end OLTP.
- App programmers view their data as an object graph.
- ORMs are baked into popular frameworks.

Management Issues.

- Web programmers are relatively cheap.
- SQL experts are relatively expensive.
- The problems can be blamed on the RDBMS.
  - Database portability is considered good by definition.
- Profit!

Quick Fixes.

- Write custom SQL and stuff it inside of object APIs.
- Create friendly APIs for the application using stored procedures.
  - Database server as app server.
  - Plays to a serious PG strength.
- Move mass-update operations out of the OO codebase into separate processes.

Management Fixes.

- Performance issues cost money.
- An underused RDBMS makes inefficient use of the hardware.
- No real-world application is pure OLTP.
- ORMs are not a data warehousing solution.

Educational Fixes.

- `tail -f` the logs.
  - ORMs sweep a lot under the rug; take out the rug-beater.
  - Teach the relational model, not "SELECT gets the data."
  - SQL experts are expensive, remember?
    - Profit!

Management Fixes.

- Misused RDBMSes cost money.
  - Extra hardware.
  - Extra developer time.
  - Remember those expensive SQL experts?
- Get what you are paying for!

ORMs are not evil.

- They're invaluable for their core operation of object persistence.
  - We'd have to pry them out of their cold, dead hands anyway.
- Most of the problems come from the "hammer/nail" attitude.
- App programmers have been convinced that not learning SQL is a virtue.

So, how do you solve the problem?