

Everything You Wanted to Know about Enterprise Database Access

Patrick Linskey
plinskey@gmail.com

RDBs in the Enterprise

- Basics
 - Data Storage
 - Concurrent Access
 - Promises of Transactionality

RDBs in the Enterprise

- Interesting Stuff
 - Ubiquitous Access
 - Open Data Format
 - Integration Engine
 - Externally Tunable

RDBs in the Enterprise

- Bad Stuff
 - Schema evolution is a pain
 - DB server adds complexities
 - Conversion adds overhead
 - Language mismatch

Relations

- From *relational theory*
- Nothing to do with *relationships*
 - relation == table
 - relationship == join

Common Access Techniques

The Transaction Script

The Object Model

The Java Way

- Transaction Script: **JDBC**
- Object Model: **JPA**

JDBC

```
Connection c = ...;  
PreparedStatement ps = c.prepareStatement(  
    "select * from person where lname = ?");  
ps.setString(1, "Linskey");  
ResultSet rs = s.execute();
```

JPA

@Entity

```
public class Person {  
    @Id private String firstName;  
    @Id private String lastName;  
  
    public Person(String fname, String lname) {  
        firstName = fname;  
        lastName = lname;  
    }  
}
```

JPA

```
Person p = new Person("Patrick", "Linskey");  
EntityManager em = ...;  
em.persist(p);  
em.close();
```

JPA Implementations

- Statement batching
- Statement re-ordering
- Distributed transactions
- Access pattern tuning
- Large query support
- Large collections
- Flexible inheritance
- Non-relational support

.NET (C#)

- Just like Java
 - Transaction Script: **ADO.NET**
 - Object Model: **Entities & Third-party ORM** (NHibernate etc.)
- ... but better
 - **LINQ**

C# + LINQ

- **L**anguage **I**ntegrated **N**ative **Q**ueries
- Decouples the query from the data store
- Can be used with ADO.NET, entities, collections, third-party ORM, XML, ...

C# + LINQ

```
var source = ...; // IQueryable or IEnumerable
var results = from p in source
               where p.FirstName == "Patrick"
               select p;
foreach(var person in results)
    Console.WriteLine(p.LastName);
```

LINQ/SQL

```
DataContext db = new DataContext("...");  
var source = db.GetTable<Customer>();  
var results = from p in source  
               where ... select p
```

- Even for LINQ/SQL, objects must exist
- LINQ/SQL is on the ORM continuum
- LINQ/Entities: more ORM features

LINQ/SQL

```
DataContext db = new DataContext("...");  
var source = db.GetTable<Customer>();  
var results = from p in source  
               where ... select p
```

- Even for LINQ/SQL, objects must exist
- LINQ/SQL is on the ORM continuum
- LINQ/Entities: more ORM features

LINQ/Collections

```
List<Person> people = ...;  
var results = from p in people  
              where ... select p
```

- Can query over **any** collection
- Uses reflection to dig into the object model
- Trivially filter / sort in-memory collections

Ruby on Rails

- Transaction Script: **DBI**
- Object Model: **ActiveRecord**
- Optimized for rapid development
- Still missing big-E features

ActiveRecord

```
class CreatePerson <
  ActiveRecord::Migration

  def self.up
    create_table :person do |t|
      t.column :first_name, :string
      t.column :last_name, :string
    end
  end

  def self.down
    drop_table :person
  end
end
```

ActiveRecord

```
class Person < ActiveRecord::Base  
end
```

```
person = Person.new  
person.first_name = "Patrick"  
person.last_name = "Linskey"  
person.save
```

ActiveRecord

- DB-centric
 - Fields are materialized from DB metadata
- Missing big-E features
 - Collections cannot be accessed lazily
 - Roll-your-own statement batching
 - XA
- ... but DBI cohabitation is simple

Ruby off Rails

- **Sequel:** DBI and ActiveRecord alternative
- **DataMapper:** ActiveRecord alternative

Python

- Transaction Scripts: **DB-API**
- Frameworks provide varying degrees of higher-level functionality

Python + DB-API

- PEP-249: DB-API 2.0
- Statement batching: executemany
- SQL strings vary among drivers
- Optional XA support

Parameters in DB-API

MySQL, DB2: 'format'

```
cursor.execute("""  
    select * from people  
    where fname = %s and lname = %s  
""", ('Patrick', 'Linskey'))
```

Parameters in DB-API

Oracle: 'numeric' / 'named'

```
cursor.execute("""  
    select * from people  
    where fname = :1 and lname = :2  
""", ('Patrick', 'Linskey'))
```

Parameters in DB-API

SQLServer, Sybase: 'qmark'

```
cursor.execute("""  
    select * from people  
    where fname = ? and lname = ?  
""", ('Patrick', 'Linskey'))
```

Python + Django

- Object Model: Django **Models**
- Alternatives: **SQLAlchemy, Elixir**
- **dmigrations** facilitate schema evolution
- Driver-specific SQL is still a problem
- No statement batching, lazy collections

Django Models

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)

p = Person(first_name="Patrick",
            last_name="Linskey")
p.save()
```

JRuby; Jython

- Run on the JVM
 - Threading, JIT
- JDBC bindings
- JNDI Connection lookups

Alternatives

- Remember those downsides?
 - Schema evolution is a pain
 - DB server adds complexities
 - Conversion adds overhead
 - Language mismatch

Non-Relational DB

- Might be better tuned to your requirements
- Any type of DB process shares the same high-level issues outlined earlier
 - ... usually without many of the benefits

“Post-Relational”

- Column Stores
- MapReduce
 - on SQL or flat files / binary files
- MapReduce vs. Large-Scale Relational
 - Pavlo, Paulson, Rasin, Abadi, Dewitt, Madden, and Stonebraker
 - <http://tinyurl.com/cohhm3>

Filesystem

- Language-provided serialization
 - Nice integration
 - No queries (except .NET)
 - Forward compatibility takes planning
 - Roll-your-own concurrent access
 - Roll-your-own transactions (eek!)

Filesystem

- JSON as a serialization format
 - All* languages have JSON bindings
 - More expensive than serialization, but much more compatible
 - Beware the text-editing end user!

Questions?

Patrick Linskey
plinskey@gmail.com