Programming Languages & Databases: What's the problem? & Safe Query Objects

William R. Cook
UT Austin
Department of Computer Sciences

Founder & Former CTO, Allegis Corporation (now Click Commerce)

An Ancient Problem...

- **"Impedance mismatch" (David Maier, 1987)**
  - Identified syndrome, no diagnosis or cure
- **What is it really**
  - Types?
  - Relational vs. Object-Oriented?
  - Declarative vs. Procedural?
  - All of the above, or something else?
- **Let's try to pin some of it down...**
  - Focus on queries, not transactions
  - Dialog between PL and DB viewpoints

Issue #1: Search Optimization

Find employees

whose

name matches a prefix

&

salary is greater than base salary

PL Solution

- **Orthogonal Persistence**

```
void printInfo(String prefix, int base) {
  for (Employee emp : DB.getEmployees() )
    if ( emp.name.startsWith(prefix)
         && emp.salary >= base )
      print( emp.name );
}
```

- **Orthogonal Persistence**
  - Automatically load objects as needed
  - Approximations
    - Java Data Objects (JDO), EJB, Hibernate…

PL Solution?

- **Orthogonal Persistence**

```
void printInfo(String prefix, int base) {
  for (Employee emp : DB.getEmployees() )
    if ( emp.name.startsWith(prefix)
         && emp.salary >= base )
      print( emp.name );
}
```

Linear search

Rewrite the program to use an index?

DB viewpoint

- **This is crazy!**
- **Databases already do this well...**
  - Choose algorithm (plan) based on
    - Structure of the query
    - Statistical properties of data
  - Programmer shouldn't do this manually
- **How to send the *criteria* to the database?**

```
emp.name.startsWith(prefix)
    && emp.salary >= base
```


A Pragmatic Solution?

- **Criteria in strings w/parameters (JDO style)**

```
void printInfo(String prefix, int base) {
  Command q = new Query(Employee.class);
  String paramDecl = "String prefix, int base";
  String filter = "emp.name.startsWith(prefix)"
                  + " && emp.salary >= base";
  q.declareParameters( paramDecl );
  q.setFilter( filter );
  for ( Employee emp : q.execute(prefix, base) )
    print( emp.name );
}
```

- **Complex dependencies between strings/API**

PL viewpoint

- **This is crazy!**
  - Declarations in strings?
    - q.declareParameters("String prefix, int base");
  - Generically typed parameters
    - Object execute(Object p1, Object p2, ...);
- **Programming languages do this better**
  - Why reinvent the wheel?
  - No alternative using existing tools

Data Access Libraries

- **No static checking**
  - Syntax, types, parameters
- **Hard to maintain, reuse, secure**
  - Not just "boilerplate" in real apps
  - Queries often contain a lot of program's behavior
- **We need a solution that provides**
  - Performance
    - high throughput, low latency, scalability, …
  - Good software engineering
    - safety, adaptability, modularity, reuse, …

Safe Query Objects

- **Represent queries as classes**
  - Localize query behavior into query classes
  - Use ordinary PL methods for criteria
  - Provides static typing
  - Complements JDO, EJB, Hibernate
- **Provide an alternative evaluation model**
  - Compiler generates SQL + wrapper code
  - Use existing data access libraries: JDO and JDBC
    - JDO handles mapping objects to tables
    - JDBC version generates necessary loading code

Safe Query

```
class FindEmployees instantiates RemoteQuery {
   // parameters are member variables
  String prefix;
  int base;
   // filter method evaluates criteria
  boolean filter(Employee emp) {
   return emp.name.startsWith(prefix)
              && emp.salary >= base ;
  }
}
```

- Note: constructor created automatically

Using Safe Queries

- **Remote Execution**

```
Query<Employee> q =
    new FindEmployees("C", 10000);

for ( Employee emp : q.execute(db) )
   print( emp.name );
```

- **Local Execution**
  - Since query is simply a class, it can also be executed against in-memory collections
  - Query objects must be purely functional

Implementing Safe Queries

- **Prototype uses OpenJava**
  - Compile-time metaprogramming for Java
  - User-defined *metaclasses* run within the compiler
    class MyQuery instantiates **RemoteQuery**
  - Can modify or extend the program
  - A macro package for Java
- **Alternatives**
  - Exploring IL/byte-code transformation

Issue #2: Dynamic Queries

- **What if the base salary test is optional?**

```
void printInfo(String prefix, Integer base) {
  Command q = new Command(Employee.class);
  String paramDecl = "String prefix, int base";
  String filter = "emp.name.startsWith(prefix)";
  if (base != null) {
   filter += " && emp.salary >= base";
  }
  q.declareParameters( paramDecl );
  q.setFilter( filter );
  ResultSet r = q.execute(prefix, base);
  …
```


Safe Dynamic Queries

- **Use short-circuit and/or**

```
class FindEmployees instantiates RemoteQuery {
  String prefix;
  Integer base;
  boolean filter(Employee emp) {
   return emp.name.startsWith(prefix)
         && (base == null || emp.salary >= base);
  }
}
```

- **Note**
  - "base == null" does not depend upon database
  - Different queries generated if base is/is not null

More Complex Queries

- **Sorting**

```
Sort order(Employee emp) {
   return new Sort(emp.salary, Sort.DESC,
          new Sort(emp.name, Sort.ASC)));
```

- **Relationships/Joins**

```
return emp.salary > emp.manager.salary;
```

- **Existentials/Reuse**

```
boolean filter(Department dept) {
  return dept.name.startsWith(deptPrefix)
   && exists(dept.employees,
             new FindEmployees(null, base) );
```

Issue #3: Null Values

- **PLs and DBs use null differently**

	DB	PL
Operations	return unknown	throw exceptions
Usage	any type	only pointer types
And/Or	3-valued logic	throw exceptions

- **Unification**
  - Define nullable-operators (plus, times, eq, etc)
    - Return null when null passed as argument
  - Nullable data abstractions
    - Overload standard operators
  - Traversal through null returns null
    - Not all PLs provide necessary abstractions

Evaluation

- **Handles all query formats in JDO 1.0**
- **Compiere Examination**
  - A large open-source ERP application
  - Examined kinds of queries
    1. Almost all queries have parameters
    2. Many queries are dynamic
       Mostly in criteria, sometimes in sorting, rarely output
    3. Data driven queries from static data
    4. Data driven queries from dynamic data
       Query criteria stored in the database!
  - All but #4 can be handled by Safe Queries

Compiere Example Query

```
String sql = "SELECT * FROM C_Order o WHERE o.IsSOTrx='Y'";
if (p_C_Order_ID != 0)
  sql += " AND o.C_Order_ID=?";
else {
  if (p_C_BPartner_ID != 0)
    sql += " AND o.C_BPartner_ID=?";
  if (p_Vendor_ID != 0)
    sql += " AND EXISTS (SELECT * FROM C_OrderLine
                  ... WHERE po.C_BPartner_ID=?)";
  if (p_DateOrdered_From != null)
    sql += "AND TRUNC(o.DateOrdered) >= ?";
  if (p_DateOrdered_To != null)
    sql += "AND TRUNC(o.DateOrdered) <= ?";
}
```


Related Work

- **Gould, Su & Devanbu (ICSE 2004)**
  - Typechecks dynamic SQL inside strings
  - Pro: works with existing programs
  - Con: approximate; issues with modularity
- **DBPL/Tycoon (mid '90s)**
  - Language, compiler, runtime, database, remoting
    - Proposed integrated compiler and database optimizations
    - Gateway to SQL
- **Other**
  - Distributed query optimization

Related Work

- **Cw: queries for C# (Bierman et al, 2005)**
  - Remote operations delayed, sent to database

```
rows = select * from DB.Employees where City == city
foreach ( row in rows ) {
  Console.Write( row.LastName.Value + " : ");
  Console.WriteLine( row.EmployeeID.Value );
}
```

  - In progress (not full orthogonal persistence?)
- **AppleScript (Cook, 1993)**
  - Generalized queries for remote objects

```
first character of every word whose style is bold
```

  - Also supports bulk updates

Related Work

- **SQL DOM (McClure & Kruger, ICSE 2005)**

```
public String GetCustomers(String name) {
  CustomersTblSelectSQLStmt sql =
    new CustomersTblSelectSQLStmt();
  if (name != null)
    sql.AddWhereCondition(
          new CompanyNameWhereCond(name));
  return sql.GetSQL();
}
```

- **Safe Query Objects version of same query**
  - No performance impact (compile-time translation)

```
class GetCustomers instantiates RemoteQuery {
  String name;
  boolean filter(Customer cust) {
    return (name == null) || (cust.name == name);
}}
```


Conclusion

- **Safe Queries**
  - Queries are "Just Java"*
  - Statically typed
    - Queries are just normal classes
  - Remotely executed
    *Translated to SQL at compile time
  - Works well with JDO, EJB, Hibernate, JDBC, …
- **More to be done**
  - Aggregation & grouping
  - Optimizing navigation from search results
  - Extract safe queries from procedural code
  - Issues: transactions, updates, caching, …