Queryll: Java database Queries Through Bytecode Rewriting

Introduction-
- Middleware system
- Interfacing java with SQL databases by providing database query facilities to java
- Standard java syntax; no special compiler IDE
- Queryll bytecode rewriter replaces the code with equivalent SQL queries

Related Work
- Other middleware systems use special programming languages to interface java to other programming languages. All require the use of custom languages to access their features.
- Weak points: have to learn new language; have to write code between models; have to embed queries in strings that are not error-checked; have to marshal parameters into special data structures.

Related Work-
- Hybrid programming language, SQLJ
- Strong points: error checking, automatic data marshaling
- Weak points: new compiler, new IDE; not applicable with multiple interface languages; tightly bound to SQL table-oriented view of data.

Related Work
- Standard database middleware layer JDBC
- Queries in strings passed through API
- Weak points: must manually pack parameters into queries and manually read out and interpret individual fields from results; SQL table-oriented view of data

Related Work-
- ORM Tools (Hibernate, EJB)
- Strong points: specify a mapping from SQL tables to an object representation; no data marshaling issue
- Weak points: can not be used for complex queries
Related Work
- LINQ
- Support lambda expressions

Goal
- No new language, just API
- No new compilers, no data marshaling, no embedding code inside strings
- Use bytecode rewriting

Bytecode rewriting and decompiling
- Compilers compile java into bytecode, stored in classfiles, which can be executed using a Java VM
- Other bytecode rewriting techniques: JOrchestra, aspect-oriented programming tools, some ORM tools
  - Weak point: only modifying surface features of the code
- Queryll borrows techniques from classfile decompilation

Queries with Queryll
- Designed to conform with standard Java patterns for working collections
  - Cannot convert arbitrary Java to SQL
- Use ORM to allow database entities to be represented and manipulated as objects in JAVA
- Support selections, projections, joins
- No support for aggregation or nested queries; no support for SQL ordering and limit operation

Queryll ORM
- Use a custom light-weight ORM tool to map tables to classes
- Generate the classes for each entity with accessor methods for fields and special methods for traversing relationships between objects
- Create a Entity manager responsible for ensuring the database data and in-memory object representations remain consistent.

Simple Queries and Selection
- For-each loop over collections called QuerySet
- Loop = query; execution fills an output QuerySet with the results
- Loop must iterate over all original QuerySet elements, and only add elements to the new QuerySet
Example
- QuerySet<String> canadian = new QuerySet<String>;
- String country = "Canada";
- For (Client c: em.allClient())
  - If (c.getCountry().equals(country))
    - Canadian.add(c.getName());

Example
1: Br1 = r1.OfficeManager; set allOffice();
2: cf = Br1.Get; 
3: goto lab1;
lab1: 4: Bp3 = r1.Office; Object mem[];
5: if (r1.getName().equals("Seattle"))
   - Sk1 = Sk1.Office; bool b1;
   - if (b1 == 0) goto lab2;
6: if (Bp3 == 0) goto label1;
7: if (r1.getName().equals("LA"))
   - Sk1 = Sk1.Office; bool b1;
   - if (b1 == 0) goto lab2;
lab2: 8: Br0 = r1.character; boolean b2;
9: if (b2 == 0) goto label1;

Projection-
- Pair object
- QuerySet<PairAccount, Double> overstay = new QuerySet<PairAccount, Double>();
- for (Account a: em.allAccount())
  - if (a.getBalance() < a.getBalance())
    - double penalty = (a.getBalance() - a.getBalance()) * 0.001;
    - overstay.add(PairAccount, Double(a, penalty));

Join
- QuerySet<PairClient, Account>> 
  - swiss1 = new QuerySet<PairClient, Account>();
  - swiss2 = new QuerySet<PairClient, Account>();
- for (Account a: em.allAccount())
  - if (a.getHolder().getCountry().equals("Switzerland"))
    - swiss1.addPairofClient, Account(a.getHolder(), a);
- for (Client c: em.allClient())
  - if (c.getCountry().equals("Switzerland")
    - swiss2.addAll(PairCollection(c, c.getAccounts()));

Implementation-
- Labelled @Query annotation
- use Sable’s Soot framework Jimple code (execution stack)
  - for (Office of: em.allOffice())
    - if (of.getName().equals("Seattle")
      - westcoast.add(of);
    - else if (of.getName().equals("LA")
      - westcoast.add(of);

Second step-
- Identify loops within the code
- Only goto statements to describe control flow. Two ways
- Analyze as a whole; restructure back to loops
  - Use standard graph algorithms to identify loops
Third Step
- For a for-each loop to be labeled as a candidate query:
  - It must iterate over all elements; no elements left behind
  - No side effects but adding elements to another collection

Fourth Step
- Find out what sort of query it is
- Need straight-line code
- Break loops down into straight paths

Example

<table>
<thead>
<tr>
<th>Path 1</th>
<th>Path 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>15: $r7 = r6 hasNext();</td>
<td>15: $r7 = r6.hasNext();</td>
</tr>
<tr>
<td>16: if ($r7 != 0 goto label1</td>
<td>16: if ($r7 != 0 goto label1</td>
</tr>
<tr>
<td>4: $r13 = r6.next();</td>
<td>4: $r13 = r6.next();</td>
</tr>
<tr>
<td>5: $r14 = (Office) $r13;</td>
<td>5: $r14 = (Office) $r13;</td>
</tr>
<tr>
<td>6: $r15 = r14.getName();</td>
<td>6: $r15 = r14.getName();</td>
</tr>
<tr>
<td>7: $r3 = $r15.equals(&quot;Seattle&quot;);</td>
<td>7: $r3 = $r15.equals(&quot;Seattle&quot;);</td>
</tr>
<tr>
<td>8: if ($r3 == 0 goto label2</td>
<td>8: if ($r3 == 0 goto label2</td>
</tr>
<tr>
<td>(branch not taken)</td>
<td>(branch taken)</td>
</tr>
<tr>
<td>9: r11.addEdge($r4);</td>
<td>10: r16 = r14.getName();</td>
</tr>
<tr>
<td>11: $r5 = $r16.equals(&quot;LA&quot;);</td>
<td>12: $r5 = $r16.equals(&quot;LA&quot;);</td>
</tr>
<tr>
<td>13: if ($r5 == 0 goto label3</td>
<td>13: if ($r5 == 0 goto label3</td>
</tr>
<tr>
<td>(branch not taken)</td>
<td>(branch not taken)</td>
</tr>
<tr>
<td>14: r11.addEdge($r4);</td>
<td>14: r11.addEdge($r4);</td>
</tr>
</tbody>
</table>

Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial</td>
<td>$f3 = 0 AND $f5 = 0</td>
</tr>
<tr>
<td>14: r11.addEdge($r4)</td>
<td>$f3 = 0 AND $f5 = 0 AND $f16 = &quot;LA&quot;</td>
</tr>
<tr>
<td>12: $f5 = $f16.equals(&quot;LA&quot;);</td>
<td>$f3 = 0 AND $f5 = 0 AND $r4.Name = &quot;LA&quot;</td>
</tr>
<tr>
<td>11: $r16 = r14.getName();</td>
<td>$f3 = 0 AND $r4.Name = &quot;LA&quot;</td>
</tr>
<tr>
<td>8: if ($r3 == 0 goto label2</td>
<td>$f3 = 0 AND $r4.Name = &quot;LA&quot;</td>
</tr>
<tr>
<td>7: $r3 = $r15.equals(&quot;Seattle&quot;);</td>
<td>$f3 = 0 AND $r4.Name = &quot;LA&quot;</td>
</tr>
<tr>
<td>6: $r15 = r14.getName();</td>
<td>$f3 = 0 AND $r4.Name = &quot;LA&quot;</td>
</tr>
<tr>
<td>5: $r14 = (Office) $r13;</td>
<td>$f3 = 0 AND $r4.Name = &quot;LA&quot;</td>
</tr>
<tr>
<td>4: $r13 = r16.next();</td>
<td>$f3 = 0 AND $r4.Name = &quot;LA&quot;</td>
</tr>
<tr>
<td>15: $r7 = r6.hasNext();</td>
<td>$f3 = 0 AND $r4.Name = &quot;LA&quot;</td>
</tr>
<tr>
<td>Simplification</td>
<td>$f3 = 0 AND $r4.Name = &quot;LA&quot;</td>
</tr>
</tbody>
</table>

Example

```sql
SELECT ... FROM Office AS A
WHERE (((A).Name != "Seattle") AND ((A).Name = "LA"))
OR ((A).Name = "Seattle")
```
Sceptical Point

- getRelated on p.215 and doGetRelated on p.216 are not equivalent
- Java programmer is forced to write in certain ways—sometimes that's more confusing than learning a new language
- The resulting query are small and simple; too many constructs are unsupported

The end

Thank you!

wu cong
2010.02.09