Fast High Volume Reporting

Lessons Learned From the Collation-Script Solution
The Ideal Reporting Platform

- Flexible Report Design
  - It should be able to do *anything*!

- Rapid Development
  - Make it easy to code and easy to maintain.

- High Performance
  - That means limited DB hits, pipelining and parallelization.

- How Do I Build It?
XML from the Database

- Because it’s SOA compatible
- Because it works with publishing tools like XSLT/FO and Dopefo
- The real reason
Structure

- SQL result sets are tabular.
- Reports are most often hierarchical or have a nested structure.
- Even reports that use tables often organize the tables into hierarchies.
- A simple example: Imagine a town ...
The SQL results looks like this:

<table>
<thead>
<tr>
<th>School</th>
<th>Name</th>
<th>Subject</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valley High</td>
<td>Jack Johnson</td>
<td>Reading</td>
<td>B</td>
</tr>
<tr>
<td>Valley High</td>
<td>Jack Johnson</td>
<td>Writing</td>
<td>B</td>
</tr>
<tr>
<td>Valley High</td>
<td>Jack Johnson</td>
<td>Arithmetic</td>
<td>C</td>
</tr>
<tr>
<td>Valley High</td>
<td>Sam Samuels</td>
<td>Reading</td>
<td>A</td>
</tr>
<tr>
<td>Valley High</td>
<td>Sam Samuels</td>
<td>Writing</td>
<td>C</td>
</tr>
<tr>
<td>Valley High</td>
<td>Sam Samuels</td>
<td>Arithmetic</td>
<td>D</td>
</tr>
<tr>
<td>Central High</td>
<td>Bill Williams</td>
<td>Reading</td>
<td>B</td>
</tr>
<tr>
<td>Central High</td>
<td>Bill Williams</td>
<td>Writing</td>
<td>B</td>
</tr>
<tr>
<td>Central High</td>
<td>Bill Williams</td>
<td>Arithmetic</td>
<td>B</td>
</tr>
<tr>
<td>Central High</td>
<td>Dana Daniels</td>
<td>Reading</td>
<td>A</td>
</tr>
<tr>
<td>Central High</td>
<td>Dana Daniels</td>
<td>Writing</td>
<td>C</td>
</tr>
<tr>
<td>Central High</td>
<td>Dana Daniels</td>
<td>Arithmetic</td>
<td>B</td>
</tr>
</tbody>
</table>
But the report we want to produce looks more like this:

<table>
<thead>
<tr>
<th>High School</th>
<th>Student Name</th>
<th>Reading</th>
<th>Writing</th>
<th>Arithmetic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valley High</td>
<td>Jack Johnson</td>
<td>B</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>Sam Samuels</td>
<td>Reading A</td>
<td>Writing C</td>
<td></td>
<td>Arithme</td>
</tr>
<tr>
<td>Central High</td>
<td>Bill Williams</td>
<td>B</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Dana Daniels</td>
<td>Reading A</td>
<td>Writing C</td>
<td></td>
<td>Arithme</td>
</tr>
</tbody>
</table>
How do we create nested data?

- SQL can’t help us.
- Dozens of nested JDBC (or other DB API) calls aren’t an option.
- Caching data in the application is a waste.
This doesn’t scale.

for school in db.execute('select name, id from school'):
    print school.name

for student in db.execute('select name, id from student where school_id = %s' % school.id):
    print ' ', student.name

for grade in db.execute('select subject, letter from grade where student_id = %s' % student.id):
    print '  ', grade.subject, grade.letter
This scales better, but ... Yuck!

```python
for school in ORMSchools:
    print school.name

    for student in ORMStudents_by_school[school.id]:
        print ' ', student.name

    for grade in ORMGrades_by_student[student.id]:
        print '  ', grade.subject, grade.letter
```
The Simple Solution

- Execute this code in the database.
- If you create nested structure *in* the DB, you must return next structure *from* the DB.
- XML is the obvious answer.
<table>
<thead>
<tr>
<th>XQuery</th>
<th>Collation-Script</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>for $dist in $district/row return &lt;district&gt; &lt;name&gt;{$dist/name}&lt;/name&gt;</code></td>
<td><code>&lt;col:for-each type=&quot;district&quot;&gt; &lt;district&gt; &lt;name&gt;{name}&lt;/name&gt; &lt;/district&gt;</code></td>
</tr>
<tr>
<td><code>{ for $sch in $school/row where $sch/district_id = $dist/district_id return &lt;school&gt;{$sch/name}&lt;/school&gt; }</code></td>
<td><code>&lt;col:for-each type=&quot;school&quot;&gt; &lt;school&gt;{name}&lt;/school&gt; &lt;/col:for-each&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/district&gt;</code></td>
<td><code>&lt;/col:for-each&gt; &lt;/col:for-each&gt;</code></td>
</tr>
</tbody>
</table>
What is Collation-Script?

- It's a query language.
- It's a reporting platform.
- It's a report-stream definition language.
- But can it make julienne fries?
Structure: The Next Level

- Define where one document ends and another begins.
- Define which documents go into which files and in what order.
- Structure the fulfillment via printer workflow.
Print Vendor Support

- Shipping Units
- Slip Sheets
- Addresses and Mailing Labels
Shipping Unit 87

Central High
R.F.D. 3
Sunnyvale, RI 02893
Let’s Write a Collation-Script

<col:for-each type="district">
  <col:file path="D_{district_id}.fo" processor="dopefo.exe">
    <col:for-each type="school" orderby="school_id">
      <col:shipping-unit address="{address} {city}, {state} {zip}">
        <col:for-each type="student" orderby="last, first">
          <col:document doctype="st_report" stylesheet="sr.xsl">
            <student-name>{last}, {first}</student-name>
            <district>{district.name}</district>
            <school>{school.name}</school>
            ... More student report data ...
          </col:document>
        </col:for-each>
      </col:shipping-unit>
    </col:for-each>
  </col:file>
</col:for-each>
Let’s Write a Collation-Script

<col:for-each type="district">
    <col:file path="D_{district_id}.fo" post-processor="dopefo.exe">
        <col:for-each type="school" orderby="school_id">
            <col:shipping-unit address="{address} {city} {state} {zip}">
                <col:for-each type="student" orderby="last_name, first_name">
                    <col:document doctype="student_report" stylesheet="sr.xsl">
                        <student-name>{last_name}, {first_name}</student-name>
                        <district>{district.name}</district>
                        <school>{school.name}</school>
                        ... More student report data ...
                    </col:document>
                </col:for-each>
            </col:shipping-unit>
        </col:for-each>
    </col:file>
</col:for-each>
Performance

- XSLT is Slow
- Supercomputers Are Expensive
- Clusters Are Complicated
- Parallelize the Easy Way
Let the compiler do it.

Just rewrite this:

```xml
<col:for-each type="district">
    <col:file path="D_{district_id}.fo">
        <col:for-each type="school">
            ... etc...
        </col:for-each>
    </col:file>
</col:for-each>
```

as this:

```xml
<col:file path="D_{district_id}.fo">
    <col:for-each type="school">
        ... etc...
    </col:for-each>
</col:file>
```

and you’re halfway there.
Meta-Reporting

- Planning
- Quality Control
- Accounting
Conclusion

- XML from the DB eliminates extra code and performance bottle necks.
- Creating a complete reporting platform becomes easier when you’ve got a ‘report stream definition language’ rather than just a ‘query language’.
- Computer programming is language design.