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:

Valley High	Jack Johnson	Reading	B
Valley High	Jack Johnson	Writing	B
Valley High	Jack Johnson	Arithmetic	C
Valley High	Sam Samuels	Reading	A
Valley High	Sam Samuels	Writing	C
Valley High	Sam Samuels	Arithmetic	D
Central High	Bill Williams	Reading	B
Central High	Bill Williams	Writing	B
Central High	Bill Williams	Arithmetic	B
Central High	Dana Daniels	Reading	A
Central High	Dana Daniels	Writing	C
Central High	Dana Daniels	Arithmetic	B

But the report we want to produce looks more like this:

```
Valley High

      Jack Johnson
          Reading       B
          Writing       B
          Arithmetic    C

      Sam Samuels
          Reading       A
          Writing       C
          Arithmetic    D

  Central High

      Bill Williams
          Reading       B
          Writing       B
          Arithmetic    B

      Dana Daniels
          Reading       A
          Writing       C
          Arithmetic    B
```

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!

```
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.

XML Query Showdown

XQuery	Collation-Script

for $dist in $district/row
return
<district>
<name>{$dist/name}</name>
{
  for $sch in $school/row
  where $sch/district_id =
  $dist/district_id
  return
  <school>{$sch/name}</school>
}
</district>
	<col:for-each type="district">
	 <district>
	  <name>{name}</name>
	
	 <col:for-each type="school">
	  <school>{name}</school>
	 </col:for-each>
	
	 </district>
	</col:for-each>

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

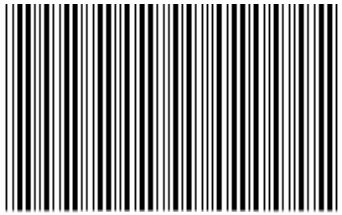
- Shipping Units
- Slip Sheets
- Addresses and Mailing Labels

Shipping Unit 87

Central High

R.F.D. 3

Sunnyvale, RI 02893



00087

Shipping Unit 87

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 ...
```

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:

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

as this:

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

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.