



Simplifying Full-Text Search Over RDBMS Data

Powerful search results using Solr
– regardless of storage method

Thinking Lucene ▼ Think Lucid.

About Today's Presenter



Nicholas Chase

Database/XML Author and Expert

Nicholas Chase is the author of several books on XML and web programming. He has been a trainer for Oracle Education, written tutorials for IBM developerWorks, and built numerous web apps. He currently works as a web programmer and writes tutorials on various technical subjects. Find him at <http://nicholaschase.com>.

Agenda

- ▼ Search and Databases
- ▼ Indexing data with Solr
- ▼ Searching using SQL vs. Solr
- ▼ Using multiple search criteria
- ▼ Creating a faceted search
- ▼ Getting Started: Resources

Slides are posted for download at the end of this presentation; full replay available within ~48 hours of live webcast



Boston, Massachusetts | Hyatt Harborside
October 5-6, 2010 | Training
October 7-8, 2010 | Conference
www.lucenerevolution.org

Full-Text Search Over RDBMS Data

- ▶ As data expands in the rate of its accumulation and the variety of its formats, **many developers turn to databases to store it all** -- including, more often than not, **unstructured, non-tabular text in documents of different shapes and sizes.**
- ▶ While many common databases such as Oracle and MySQL offer Full Text Search functionality, going beyond a simple keyword search **to provide the search experience users have come to expect can require complex programming.**
- ▶ Often, the problems of **querying text content** can be much more easily **solved** using a **search platform** such as **Apache Solr/Lucene**

Why does search matter?

Then:

- ▼ Most of the data encountered created for the web
- ▼ Heavy use of a site's search function considered a failure in navigation.

Now:

- ▼ Navigation not always relevant
- ▼ Less patience to browse
- ▼ Users are used to “navigation by search box”

The allure of the database

Pros:

- ▼ Search
- ▼ Data can be repurposed
- ▼ Single environment for backup and management

Cons:

- ▼ Not all data is ideal for a database

What users expect when they search

- ▶ **Immediate results:**
if they don't see what they're looking for immediately, they assume it's not there

- ▶ **Berrypicking:**
a little bit of information here, a little bit there

Bells and whistles can make all the difference

- ▶ Quality results, with more relevant results first
- ▶ Eliminating noise by using stop words
- ▶ Faceted searching
- ▶ Autocomplete
- ▶ Spell-check
- ▶ Related results

But who wants to build all that from scratch?

Why re-invent the wheel?

Apache Solr (the Lucene Search Server) provides:

- ▶ Quality results, with more relevant results first
- ▶ Eliminating noise by using stop words
- ▶ Faceted searching
- ▶ Autocomplete
- ▶ Spell-check
- ▶ Related results



What is Solr?

Open source Lucene (Enterprise) Search Server with

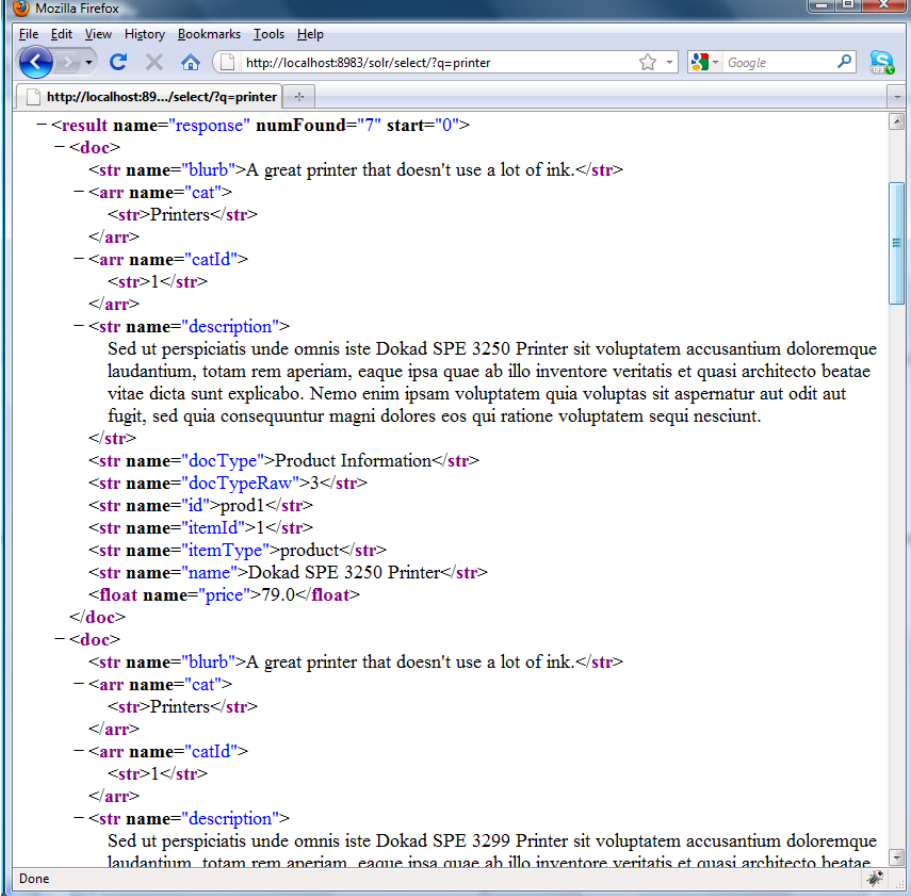
- ▶ **Advanced Full-Text Search Capabilities**
- ▶ **Optimized for High Volume, High Scale Search Traffic**
- ▶ **Standards Based Open Interfaces for extensibility**
- ▶ **Scalability through replication and sharding to multi-billion document collections**
- ▶ **Flexible and Adaptable with XML-driven configuration**
- ▶ **Extensible Plugin Architecture**
- ▶ **Deploys as Java application**



So powerful, and yet so easy

http://localhost:8983/solr/select/?q=*:*

Solr results are returned
via HTTP.



```

- <result name="response" numFound="7" start="0">
  - <doc>
    <str name="blurb">A great printer that doesn't use a lot of ink.</str>
    - <arr name="cat">
      <str>Printers</str>
    </arr>
    - <arr name="catId">
      <str>1</str>
    </arr>
    - <str name="description">
      Sed ut perspiciatis unde omnis iste Dokad SPE 3250 Printer sit voluptatem accusantium doloremque
      laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae
      vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut
      fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.
    </str>
    <str name="docType">Product Information</str>
    <str name="docTypeRaw">3</str>
    <str name="id">prod1</str>
    <str name="itemId">1</str>
    <str name="itemType">product</str>
    <str name="name">Dokad SPE 3250 Printer</str>
    <float name="price">79.0</float>
  </doc>
  - <doc>
    <str name="blurb">A great printer that doesn't use a lot of ink.</str>
    - <arr name="cat">
      <str>Printers</str>
    </arr>
    - <arr name="catId">
      <str>1</str>
    </arr>
    - <str name="description">
      Sed ut perspiciatis unde omnis iste Dokad SPE 3299 Printer sit voluptatem accusantium doloremque
      laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae

```

Enter Solr's DataImportHandler

But what about those of us married to an RDBMS...?

Solr's DataImportHandler:

- ▶ Is configurable via text files
- ▶ Indexes data right from the RDBMS
- ▶ Can be used with Solr Cell to index rich documents directly

Building the example project

- ▶ Indexing data with Solr
- ▶ Searching using SQL
- ▶ Searching using Solr
- ▶ Using multiple search criteria
- ▶ Creating a faceted search

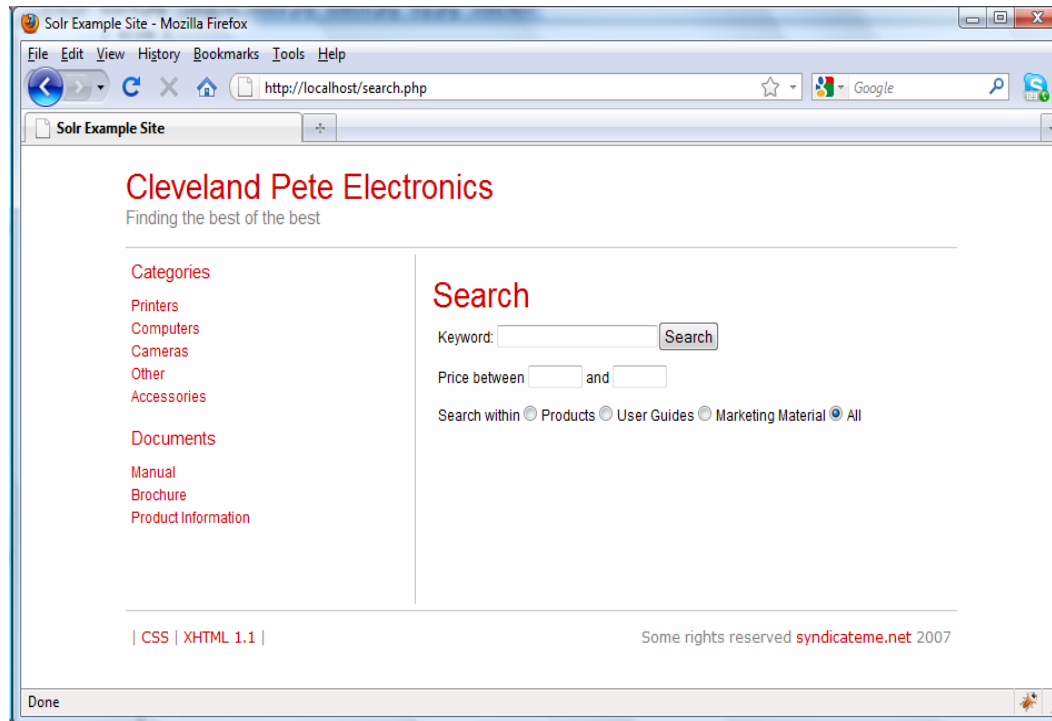
What you need to get started

To run the examples, you'll need:

- ▼ A JDBC-compliant database, such as MySQL
- ▼ A PHP-enabled web server, such as Apache
- ▼ A Lucene/Solr installation, such as Lucidworks for Solr

Concepts are language- and server-independent

The example



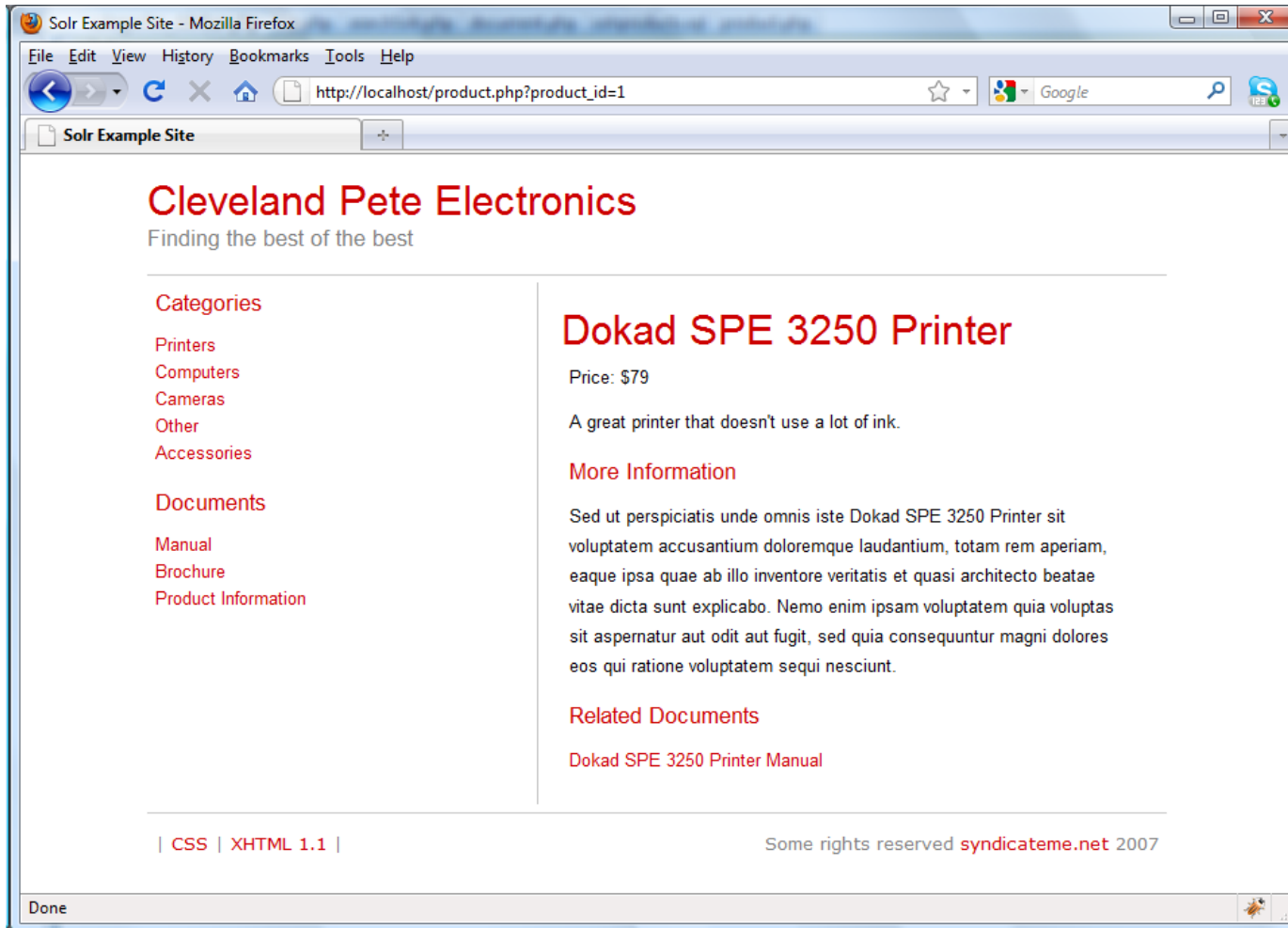
Traditional RDBMS data

Unstructured data

First via SQL

Then via Solr

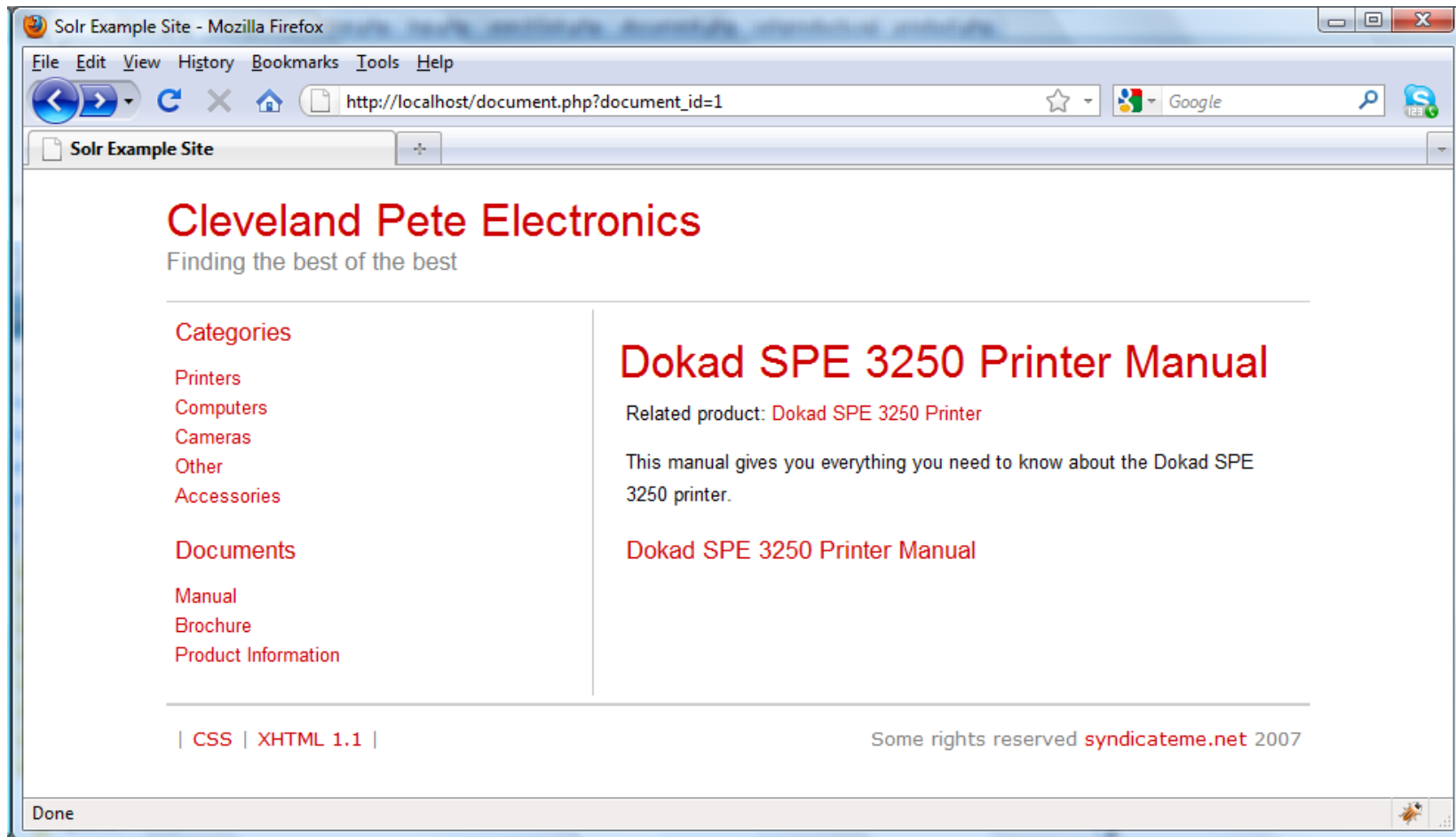
The RDBMS way: products



The RDBMS way: products

```
CREATE TABLE IF NOT EXISTS `products` (  
  `id` int(11) NOT NULL AUTO_INCREMENT,  
  `product_name` varchar(255) NOT NULL,  
  `product_price` double NOT NULL,  
  `product_adcopy` text NOT NULL,  
  `product_description` text NOT NULL,  
  PRIMARY KEY (`id`)  
)
```

The RDBMS way: documents



The RDBMS way: documents

```

CREATE TABLE IF NOT EXISTS `documents` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `cat_id` int(11) NOT NULL,
  `prod_id` int(11) NULL,
  `doc_type_id` int(11) NOT NULL,
  `document_title` varchar(255) NOT NULL,
  `document_desc` text NOT NULL,
  `document_text` text NOT NULL,
  `document_url` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
)

```

The RDBMS way: relationships

```

CREATE TABLE IF NOT EXISTS `categories` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `category_name` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
)
CREATE TABLE IF NOT EXISTS `category_products` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `cat_id` int(11) NOT NULL,
  `prod_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
)
CREATE TABLE IF NOT EXISTS `document_types` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `document_type_name` varchar(255) NOT NULL,
  PRIMARY KEY (`id`)
)

```

The Solr way: documents

Documents need to be **indexed first** so they can
then be searched

▼ **We need the data ...**

▼ **... and what that data represents.**

The Solr way: the schema

Solr provides a way to define fieldTypes and fields:

```
<field name="id" type="string" indexed="true" stored="true"
  required="true" />
<field name="itemId" type="string" indexed="true"
  stored="true"/>
<field name="name" type="text" indexed="true"
  stored="true"/>
<field name="catId" type="text_ws" indexed="true"
  stored="true" multiValued="true"/>

<uniqueKey>id</uniqueKey>

<defaultSearchField>text</defaultSearchField>

<copyField source="cat" dest="text"/>
<copyField source="name" dest="text"/>
<copyField source="manu" dest="text"/>
```

The Solr way: documents

```

<doc>
  <field name='id'>prod2</field>
  <field name='itemId'>2</field>
  <field name='itemType'>product</field>
  <field name='docType'>Product Information</field>
  <field name='docTypeRaw'>3</field>
  <field name='catId'>3</field>
  <field name='catId'>4</field>
  <field name='cat'>Cameras</field>
  <field name='cat'>Accessories</field>
  <field name='name'>Kinok UltraCam</field>
  <field name='price'>300</field>
  <field name='blurb'> Boy, this is a great camera, and it hooks up to your printer
terrifically.</field>
  <field name='description'> Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed
do eiusmod tempor incididunt ut labore et dolore magna aliqua...</field>
</doc>

```

The Solr way: how to get documents

- ▶ **Manually indexing data**
- ▶ **Extracting data with custom applications**
- ▶ **Solr Cell (Content Extraction Library – based on Apache Tika) indexes documents such as...**
 - ▶ HTML
 - ▶ XML
 - ▶ Microsoft Office
 - ▶ OpenDocument Format
 - ▶ PDF
 - ▶ RTF
 - ▶ Audio
 - ▶ Images
 - ▶ Video

The DataImportHandler

- 
First, let Solr know to where to look for the DataImportHandler. In solrconfig.xml:

```

<requestHandler name="/productimport"
  class="org.apache.solr.handler.dataimport.DataImportHandler">
  <lst name="defaults">
    <str name="config">data-config-prods.xml</str>
  </lst>
</requestHandler>

```

The data-config-prods.xml file

Next create the data-config-prods.xml file and define the datasource or sources:

```
<dataConfig>
  <dataSource
    type="JdbcDataSource"
    driver="com.mysql.jdbc.Driver"
    url="jdbc:mysql://localhost/solrProducts"
    user="solr"
    password="solr" />
  <document>
    ...
```

Example data-config-prods.xml file

```

<entity name="productsEntity"
  query="select id, id as documentId, 'product' as itemType,
'Product Information' as docType, '3' as doc_type_id, product_name,
product_price, product_adcopy, product_description from products"
  transformer="TemplateTransformer">

  <field column="documentId" name="id"
    template="prod${productsEntity.documentId}" />
  <field column="id" name="itemId" />
  <field column="itemType" name="itemType" />
  <field column="docType" name="docType" />
  <field column="doc_type_id" name="docTypeRaw" />

  <entity name="innerprod" .../>

  <field column="product_name" name="name" />
  <field column="product_price" name="price" />
  <field column="product_adcopy" name="blurb" />
  <field column="product_description" name="description" />
</entity>

```

Example data-config-prods.xml file

```

<entity name="productsEntity"
  query="select id, id as documentId, 'product' as ..."
  transformer="TemplateTransformer">
  ...
  <field column="doc_type_id" name="docTypeRaw" />

  <entity name="innerprod"
    query="select cat_id, category_name from
      category_products, categories where categories.id =
      cat_id and prod_id = ${productsEntity.id}">

    <field column="cat_id" name="catId"/>
    <field column="category_name" name="cat"/>

  </entity>

  <field column="product_name" name="name"/>
  ...
</entity>

```

Call the request handler

We defined the request handler in solrconfig.xml, so point the browser at:

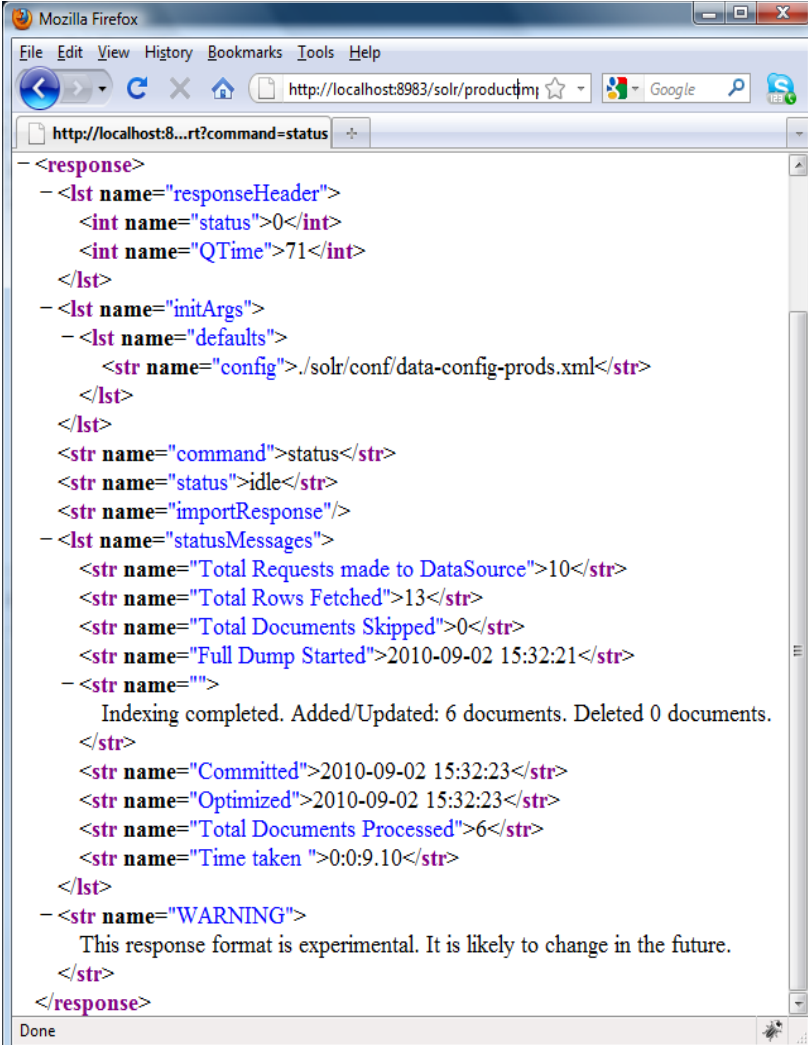
<http://localhost:8983/solr/productimport?command=full-import&clean=false>

Check the status:

<http://localhost:8983/solr/productimport?command=status>

Call the request handler

Firefox gives us a great view of the query results:



```

<response>
  <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">71</int>
  </lst>
  <lst name="initArgs">
    <lst name="defaults">
      <str name="config">./solr/conf/data-config-prods.xml</str>
    </lst>
  </lst>
  <str name="command">status</str>
  <str name="status">idle</str>
  <str name="importResponse"/>
  <lst name="statusMessages">
    <str name="Total Requests made to DataSource">10</str>
    <str name="Total Rows Fetched">13</str>
    <str name="Total Documents Skipped">0</str>
    <str name="Full Dump Started">2010-09-02 15:32:21</str>
  </lst>
  <str name="">
    Indexing completed. Added/Updated: 6 documents. Deleted 0 documents.
  </str>
  <str name="Committed">2010-09-02 15:32:23</str>
  <str name="Optimized">2010-09-02 15:32:23</str>
  <str name="Total Documents Processed">6</str>
  <str name="Time taken ">0:0:9.10</str>
  </lst>
  <str name="WARNING">
    This response format is experimental. It is likely to change in the future.
  </str>
</response>

```

Checking the import by searching

A simple “find everything” search:

`http://localhost:8983/solr/select/?q=*:*`

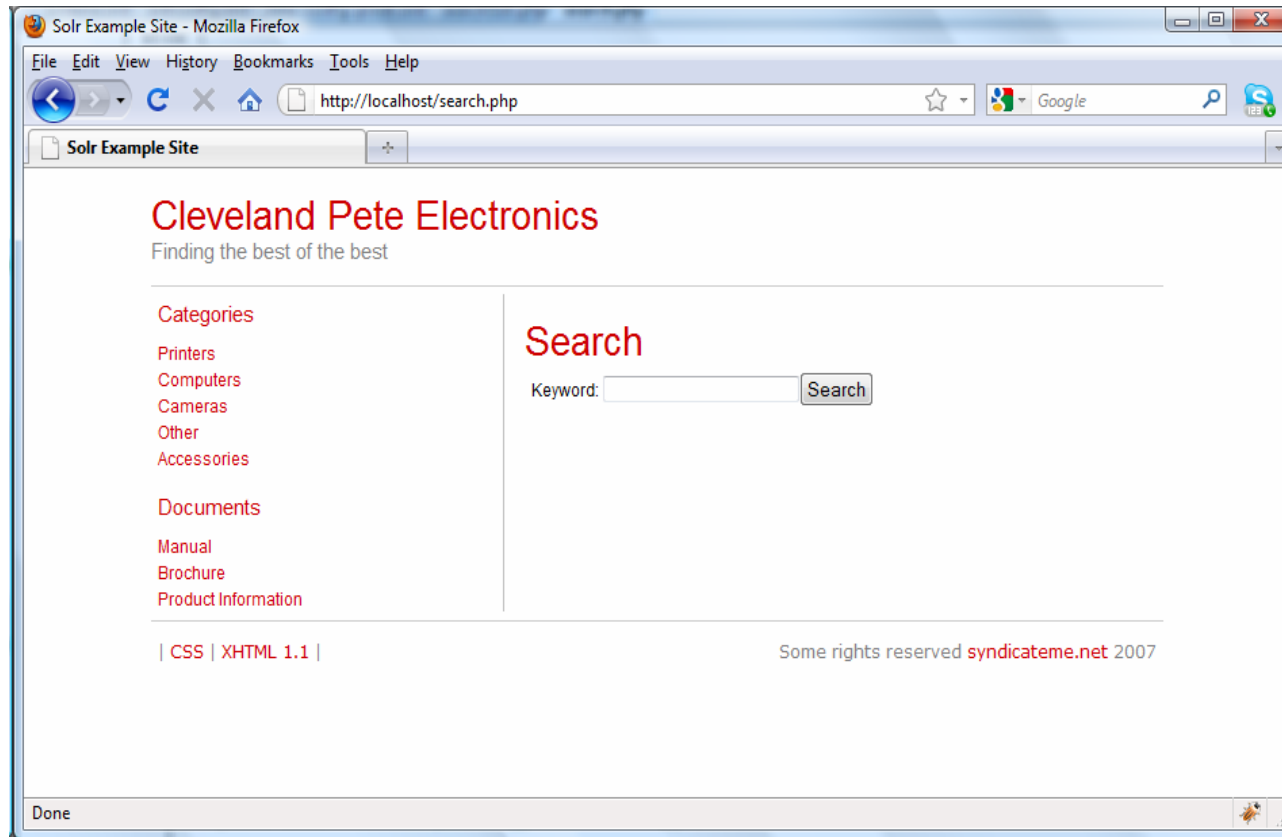


```

- <response>
  - <lst name="responseHeader">
    <int name="status">0</int>
    <int name="QTime">2</int>
    - <lst name="params">
      <str name="q">*:*</str>
    </lst>
  </lst>
  - <result name="response" numFound="8" start="0">
    - <doc>
      <str name="blurb">A great printer that doesn't use a lot of ink.</str>
      - <arr name="cat">
        <str>Printers</str>
      </arr>
      - <arr name="catId">
        <str>1</str>
      </arr>
      - <str name="description">
        Sed ut perspiciatis unde omnis iste Dokad SPE 3250 Printer sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae
      </str>
    </doc>
  </result>
</response>

```

The example project



The example project: Searching via SQL

```

if (isset($_GET['keyword'])) {
    $term = $_GET['keyword'];
    $resultsFound = false;

    $SQL = "SELECT * FROM products where (product_name
like '%" . mysql_real_escape_string ($term) . "%' or
product_adcopy like '%" . mysql_real_escape_string
($term) . "%' or product_description like
 '%" . mysql_real_escape_string ($term) . "%')";
    $results = mysql_db_query($db, $SQL, $cid);

    while ($row = mysql_fetch_array($results)) {
        $prodId = $row["id"];
        $prodName = $row["product_name"];
        $prodDesc = $row["product_description"];
        echo ("

## <a href='products.php?id=$prodId'>". $prodName . "</a></h2>"); echo ("


```

The example project: Searching via SQL

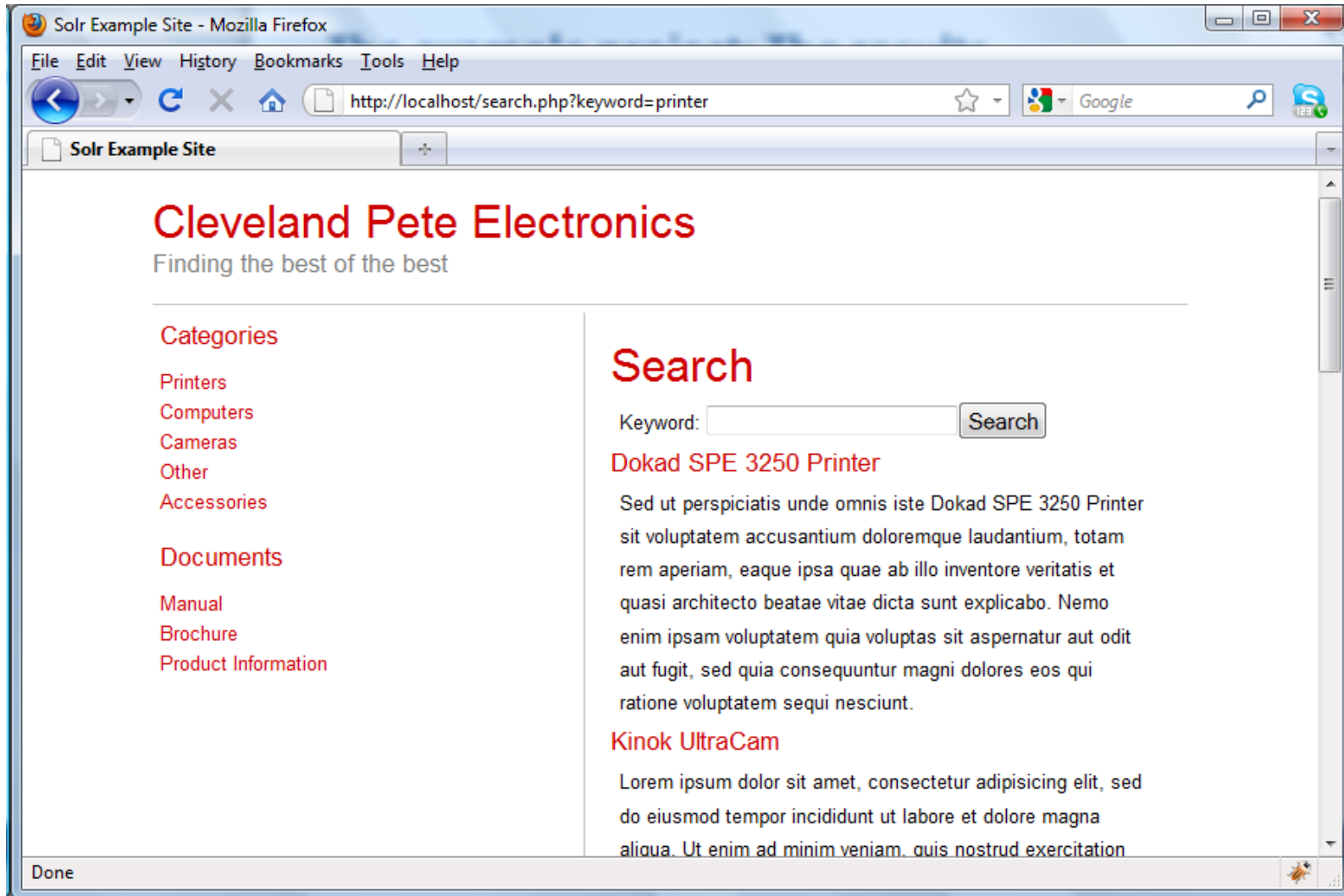
```

$SQL = "SELECT * FROM documents where (document_title like
'%" .mysql_real_escape_string($term)."% ' or document_desc like
'%" .mysql_real_escape_string($term)."% ' or document_text like
'%" .mysql_real_escape_string($term)."% ')" ;
$results = mysql_db_query($db, $SQL, $cid);

while ($row = mysql_fetch_array($results)) {
    $docId = $row["id"];
    $docTitle = $row["document_title"];
    $docDesc = $row["document_desc"];
    echo ("<h2><a href='document.php?id=$docId'>".
        $docTitle."</a></h2>");
    echo ("<p>$docDesc</p>\n");
    $resultsFound = true;
}
if (!$resultsFound){
    echo "<h2>No results found.</h2>";
}

```

The example project: The results



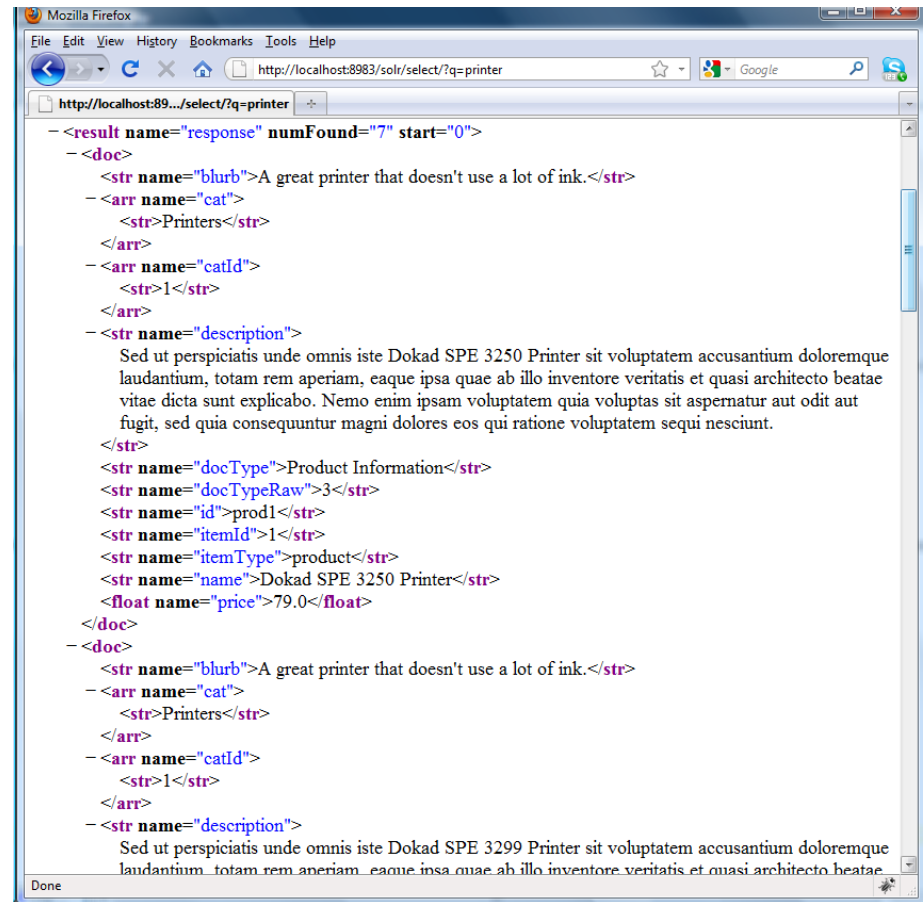
The downsides of searching via SQL

- ▼ The page works, which is nice, but it has a few limitations and complications.
 - ▼ Two different tables
 - ▼ Can't search them together
 - ▼ Hard-coding the query

The example project: searching via Solr

http://localhost:8983/solr/select/?q=*:

Solr results are returned via HTTP.



```

Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8983/solr/select/?q=printer
http://localhost:89.../select/?q=printer
- <result name="response" numFound="7" start="0">
  - <doc>
    <str name="blurb">A great printer that doesn't use a lot of ink.</str>
    - <arr name="cat">
      <str>Printers</str>
    </arr>
    - <arr name="catId">
      <str>1</str>
    </arr>
    - <str name="description">
      Sed ut perspiciatis unde omnis iste Dokad SPE 3250 Printer sit voluptatem accusantium doloremque
      laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae
      vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit aspernatur aut odit aut
      fugit, sed quia consequuntur magni dolores eos qui ratione voluptatem sequi nesciunt.
    </str>
    <str name="docType">Product Information</str>
    <str name="docTypeRaw">3</str>
    <str name="id">prod1</str>
    <str name="itemId">1</str>
    <str name="itemType">product</str>
    <str name="name">Dokad SPE 3250 Printer</str>
    <float name="price">79.0</float>
  </doc>
  - <doc>
    <str name="blurb">A great printer that doesn't use a lot of ink.</str>
    - <arr name="cat">
      <str>Printers</str>
    </arr>
    - <arr name="catId">
      <str>1</str>
    </arr>
    - <str name="description">
      Sed ut perspiciatis unde omnis iste Dokad SPE 3299 Printer sit voluptatem accusantium doloremque
      laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae
  
```

The example project: searching via Solr

```

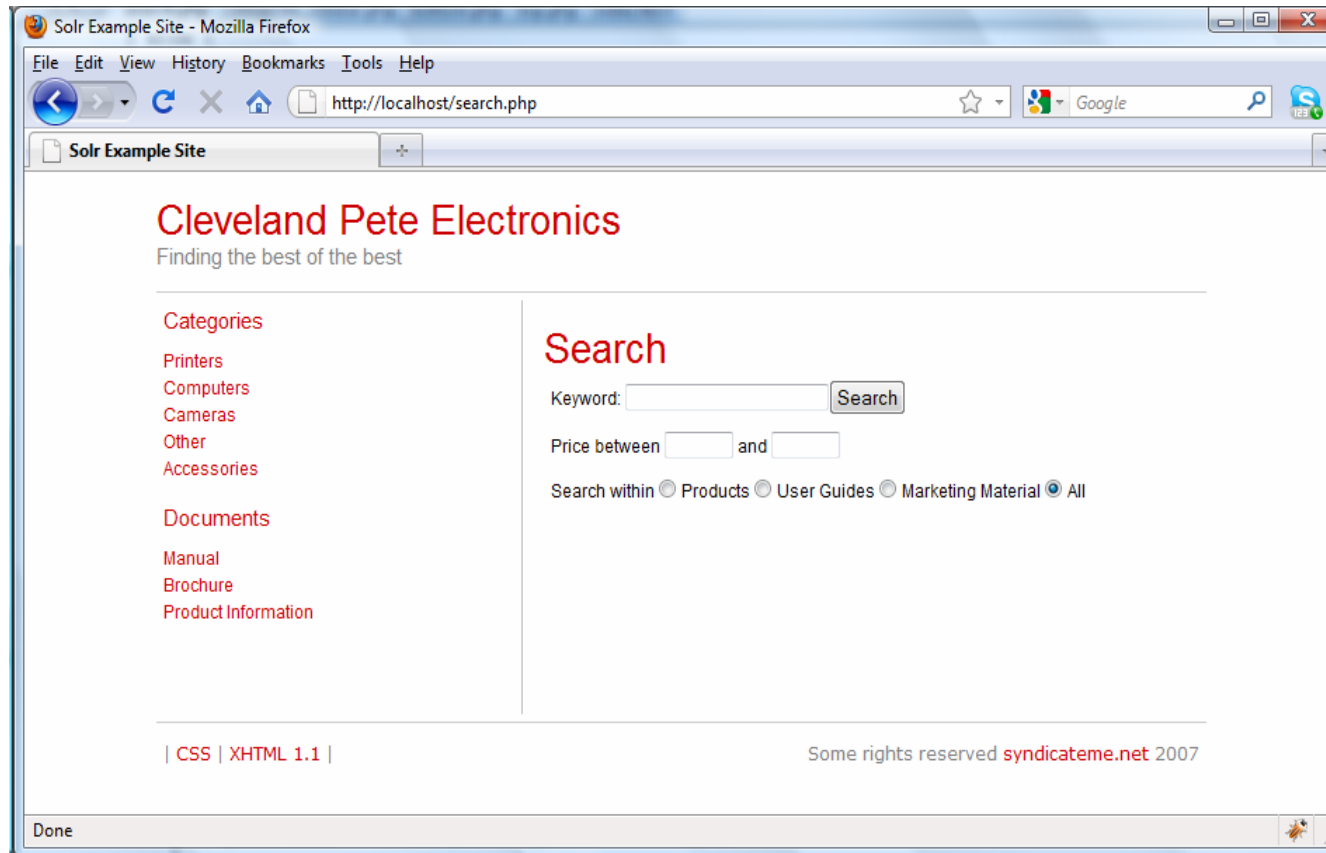
$jsonurl =
  "http://localhost:8983/solr/select/?q=".urlencode($term)."&wt=json";
$json = file_get_contents($jsonurl,0,null,null);
$json_output = json_decode($json);
$response = $json_output->response;
$resultsFound = $response->numFound;
if ($resultsFound > 0){
  echo("<h2>$resultsFound results found</h2><br />");
  foreach($response->docs as $thisDoc){
    $pageName = $thisDoc->itemType;
    $itemId = $thisDoc->itemId;
    $name = $thisDoc->name;
    $desc = $thisDoc->description;
    echo("<h2><a href='\$pageName.php?\$pageName_id=\$itemId'>".
      "$name</a></h2>");
    echo("<p>$desc</p>");
  }
} else {
  echo "<h2>No results found</h2>";
}

```

The example project: advantages of using Solr

- ▼ Display isn't dependent on the query
- ▼ Can search on multiple fields or other criteria without affecting the display
- ▼ All data in a single set (great for relevance)

The example project: using multiple fields



The example project: validating inputs

```

<script type="text/javascript">

$(document).ready(function(){

    $("#searchForm").submit(function(e){
        var lower = $("#lower").val();
        var upper = $("#upper").val();
        if (isNaN(lower) || isNaN(upper)){
            alert("Upper and lower price values "+
                "must be numeric.");
            return false;
        } else {
            return true;
        }
    });

});

</script>
  
```

The example project: adding additional fields

```
$resultsFound = false;
$priceUsed = false;

if ($docType == '3' || $docType == '*'){
    $SQL = " SELECT * FROM products where (product_name like
'%".mysql_real_escape_string ($term)."%' or ...";
    if ($lower.$upper != ''){
        $priceWhere = "";
        if ($lower != ''){
            $priceWhere .= " and product_price >= ".
                mysql_real_escape_string($lower);
        }
        if ($upper != ''){
            $priceWhere .= " and product_price <= ".
                mysql_real_escape_string($upper);
        }
        $SQL .= $priceWhere;
        $priceUsed = true;
    }
}
...
```

The example project: adding additional fields

```
if (($docType == '1' || $docType == '2' || $docType == '*')
    && !$priceUsed){

    $SQL = "SELECT * FROM documents where (document_title
like '%"mysql_real_escape_string($term)."' or
document_desc like '%"mysql_real_escape_string($term)."'
or document_text like
 '%"mysql_real_escape_string($term)."'");";

    if ($docType != '*'){
        $SQL .= " and doc_type_id =".
            " mysql_real_escape_string(".$docType.)";
    }

    ...
}
```

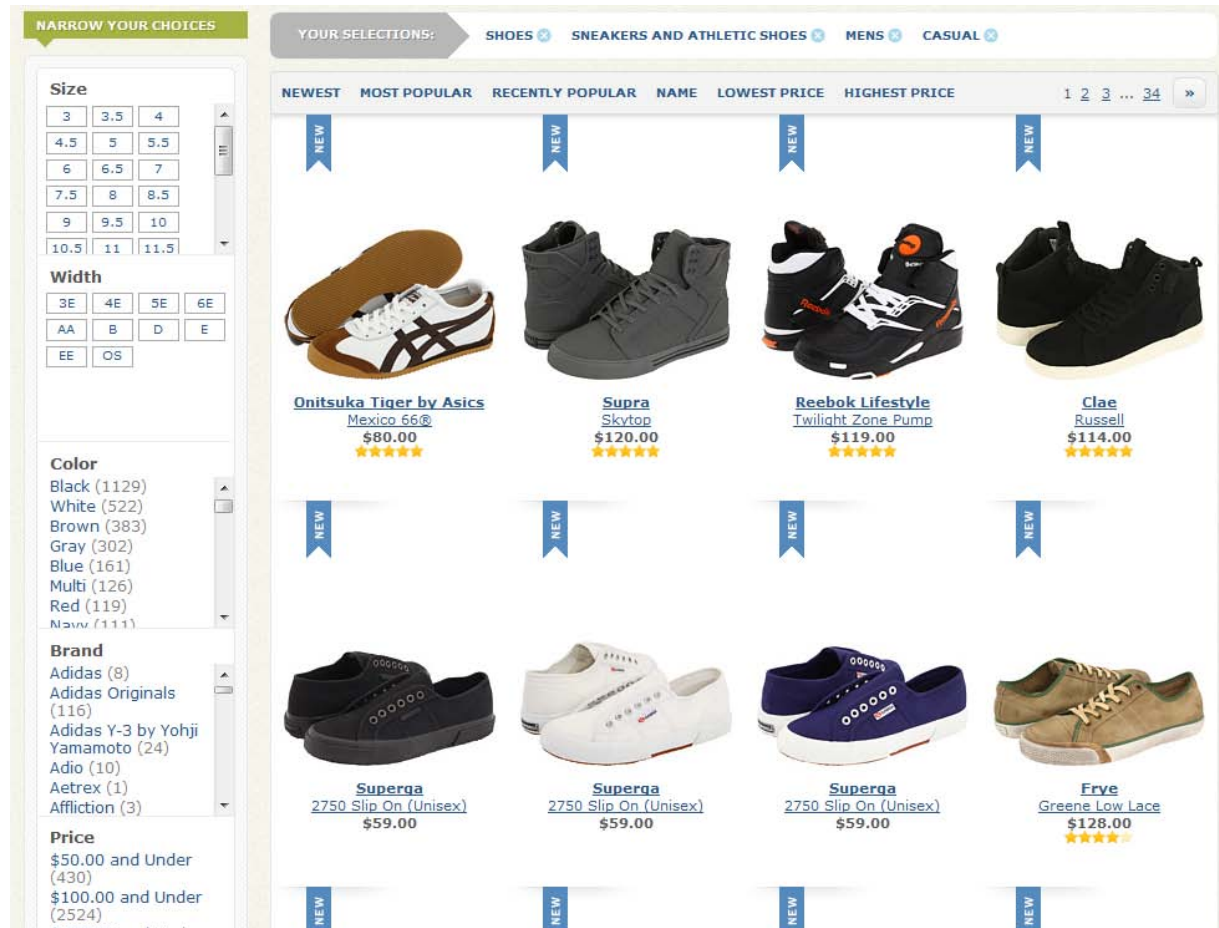
And all of this is just for two additional fields!

The example project: additional fields using Solr

```
$(document).ready(function(){
    $("#searchForm").submit(function(e){
        var lower = $("#lower").val();
        var upper = $("#upper").val();
        var keyword = $("#keyword").val();
        var docType =
            $("input[name='docType']:checked").val();

        if (isNaN(lower) || isNaN(upper)){
            alert("Upper and lower price values ...");
            return false;
        } else {
            if (keyword == "") keyword = "*";
            var query = "text:"+keyword+
                " AND docTypeRaw:"+docType+priceQuery;
            $("#keyword").val(query);
            return true;
        }
    });
});
```

Faceted Searching



Zappos.com
uses Solr
for search

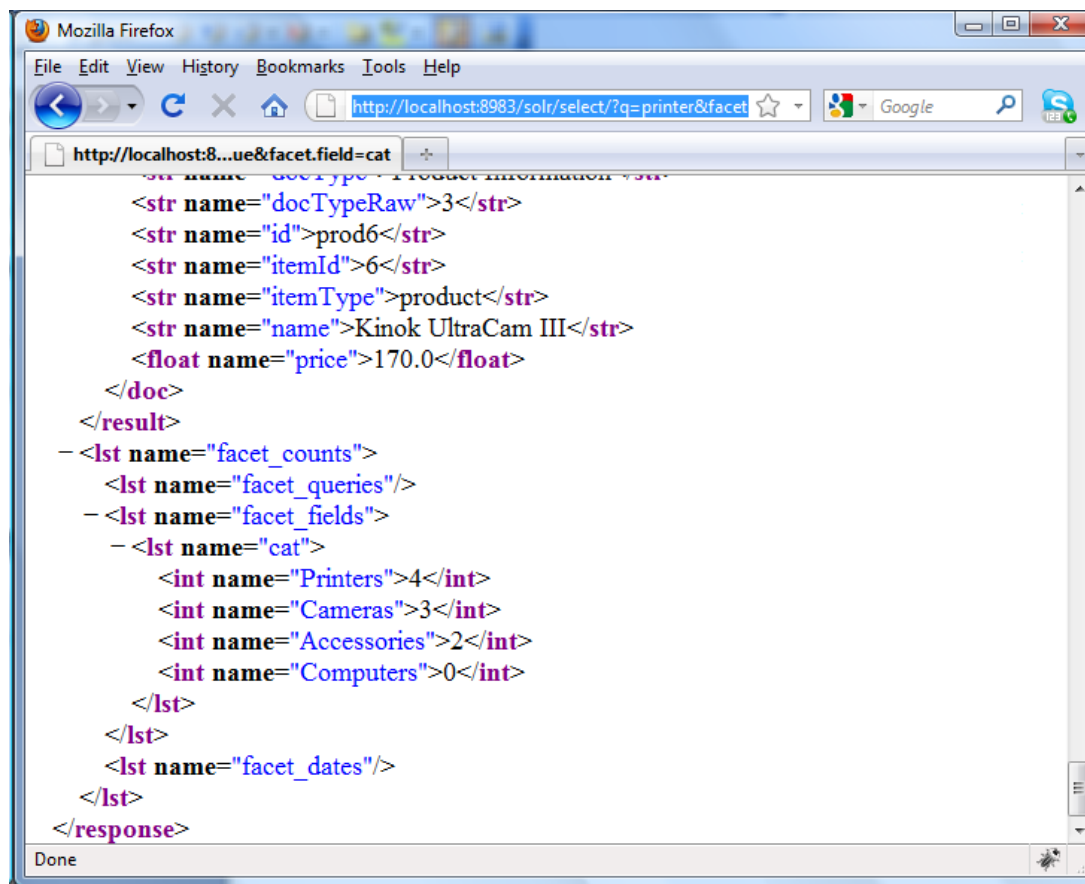
What would it take to create this using SQL?

Faceted searching: adding facets to search

[http://localhost:8983/solr/select/?q=printer&facet=true
&facet.field=cat](http://localhost:8983/solr/select/?q=printer&facet=true&facet.field=cat)

Faceting is built in
to Solr.

Adding the relevant
parameters to
the query adds
the necessary
data to the
results



```
<str name="docTypeRaw">3</str>
<str name="id">prod6</str>
<str name="itemId">6</str>
<str name="itemType">product</str>
<str name="name">Kinok UltraCam III</str>
<float name="price">170.0</float>
</doc>
</result>
- <lst name="facet_counts">
  <lst name="facet_queries"/>
  - <lst name="facet_fields">
    - <lst name="cat">
      <int name="Printers">4</int>
      <int name="Cameras">3</int>
      <int name="Accessories">2</int>
      <int name="Computers">0</int>
    </lst>
  </lst>
  <lst name="facet_dates"/>
</lst>
</response>
```

Faceted searching: adding facets to the results

```
$term = $_GET['keyword'];
$filter = "";
if ($_GET['cat'] != ""){ $filter = 'cat:'. $_GET['cat']; }
$jsonurl =
"http://localhost:8983/solr/select/?q=".urlencode($term)."&fq=" .
$filter."&facet=true&facet.field=cat&wt=json";
...
$facets = $json_output->facet_counts->facet_fields->cat;
$cats = array_chunk($facets, 2);
foreach ($cats as $thisCat){
    $catName = $thisCat[0];
    $catNumber = $thisCat[1];
    if ($catNumber > 0){
        echo "<a
href='searchSolr.php?keyword=".urlencode($term)."&cat=". $catName
.'">". $catName." (".$catNumber.)</a><br />";
    }
}
echo "<br />";
foreach($response->docs as $thisDoc){
...

```

Faceted searching: adding facets to the results

The screenshot shows a Mozilla Firefox browser window titled 'Solr Example Site'. The address bar contains the URL: `http://localhost/searchSolr.php?keyword=printer&lower=&upper=&docType=*`. The page content is as follows:

Cleveland Pete Electronics
Finding the best of the best

Categories

- Printers
- Computers
- Cameras
- Other
- Accessories

Documents

- Manual
- Brochure
- Product Information

Search

Keyword:

Price between and

Search within Products User Guides Marketing Material All

7 results found

Printers (4)

Cameras (3)

Accessories (2)

Dokad SPE 3250 Printer

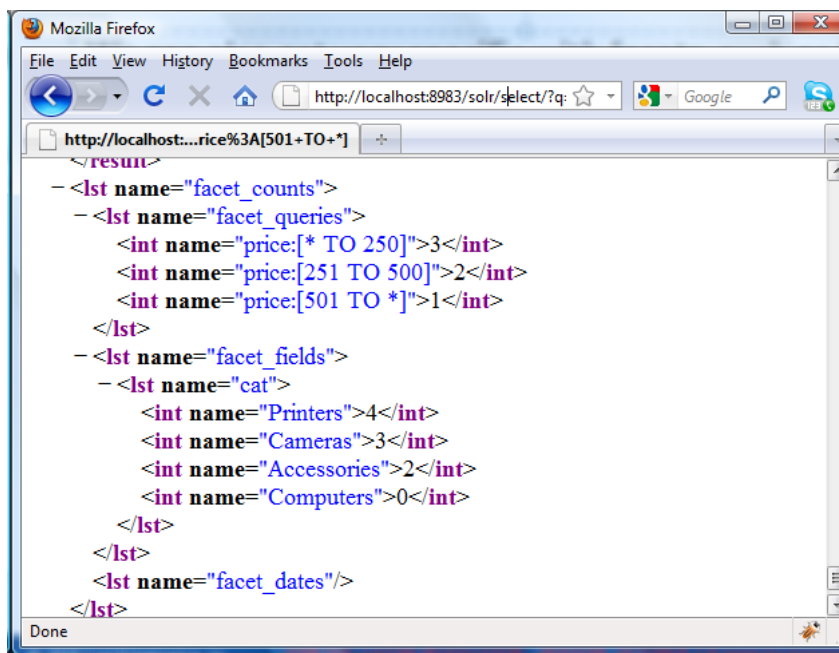
Sed ut perspiciatis unde omnis iste Dokad SPE 3250 Printer sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Nemo enim ipsam voluptatem quia voluptas sit

Done

Faceted searching: adding facets to search

We can also get very specific with facets, such as for price ranges:

```
http://localhost:8983/solr/select/?q=printer&facet=true&facet.field=cat&facet.query=price:[* TO 250]&facet.query=price:[251 TO 500]&facet.query=price:[501 TO *]
```



```
<?xml version="1.0" encoding="UTF-8" ?>
<response numFound="1" start="0">
  <result>
    - <lst name="facet_counts">
      - <lst name="facet_queries">
        <int name="price:[* TO 250]">3</int>
        <int name="price:[251 TO 500]">2</int>
        <int name="price:[501 TO *]">1</int>
      </lst>
      - <lst name="facet_fields">
        - <lst name="cat">
          <int name="Printers">4</int>
          <int name="Cameras">3</int>
          <int name="Accessories">2</int>
          <int name="Computers">0</int>
        </lst>
      </lst>
      <lst name="facet_dates"/>
    </result>
  </response>
```

Faceted searching: adding facets to the results

The screenshot shows a Mozilla Firefox browser window titled 'Solr Example Site'. The address bar contains the URL: `http://localhost/searchSolr.php?keyword=printer&lower=150&upper=300&docType=*`. The page content includes a header for 'Cleveland Pete Electronics' with the tagline 'Finding the best of the best'. On the left, there are navigation menus for 'Categories' (Printers, Computers, Cameras, Other, Accessories), 'Documents' (Manual, Brochure, Product Information), and a 'Search' section. The search section has a 'Keyword' field, a 'Search' button, a 'Price between' range selector, and radio buttons for 'Search within' (Products, User Guides, Marketing Material, All). Below the search section, it indicates '7 results found' and displays a table of facets:

Category	Price
Printers (4)	* TO 250 (3)
Cameras (3)	251 TO 500 (2)
Accessories (2)	501 TO * (1)

Below the facets, the first search result is 'Dokad SPE 3250 Printer', followed by a paragraph of placeholder text. The second result is 'Dokad SPE 3299 Printer', also followed by placeholder text. The browser status bar at the bottom shows 'Done'.

What we covered

- ▶ Indexing data with Solr
- ▶ Searching using SQL
- ▶ Searching using Solr
- ▶ Using multiple search criteria
- ▶ Creating a faceted search

What we didn't cover

Remember all these bells and whistles? Without lifting a finger, we already did:

- ▶ Quality results, with more relevant results first
- ▶ Eliminating noise by using stop words

Solr also makes it pretty easy to provide

- ▶ Autocomplete
- ▶ Spell-check
- ▶ Related results

Install LucidWorks Certified Distribution

Model your domain

Index your content

Test

Deploy



▼ Free certified distribution

- ▼ Installer
- ▼ Simple
- ▼ Plugins and enhancements
- ▼ Updateable
- ▼ Complete Reference Guide
- ▼ Support for Linux, Windows, Mac
- ▼ UI and headless both available

▼ Get started at <http://lucene.li/R>

Summary and Conclusion

- ▶ Databases are not going away as web storage
- ▶ Unstructured information is getting more common
- ▶ Solr can index and search RDBMS content pretty easily
- ▶ Solr provides the bells and whistles today's users expect
- ▶ Using Solr is easier than doing this by hand using SQL

JOIN THE LUCENE REVOLUTION—OCTOBER 7-8

THE FIRST NORTH AMERICAN CONFERENCE FOR APACHE LUCENE/SOLR



Why Attend Lucene Revolution?

- Learn from some of the leading users of Lucene/Solr including Twitter, LinkedIn, Cisco, MITRE, and Sears
- Get trained on Lucene and Solr by the experts at two-day Lucene & Solr boot camp training
- Meet the committers who write the Solr/Lucene Software
- Network with other Lucene and Solr users - and share best practices
- Meet with commercial vendors offering Lucene & Solr products and services and see how they might deliver value for your search application development and support



Boston, Massachusetts | Hyatt Harborside
October 5-6, 2010 | Training
October 7-8, 2010 | Conference

www.lucenerevolution.org

Slides are available for download
at <http://bit.ly/Solr-RDBMS>
Full replay available within
~48 hours of live webcast

Q&A

