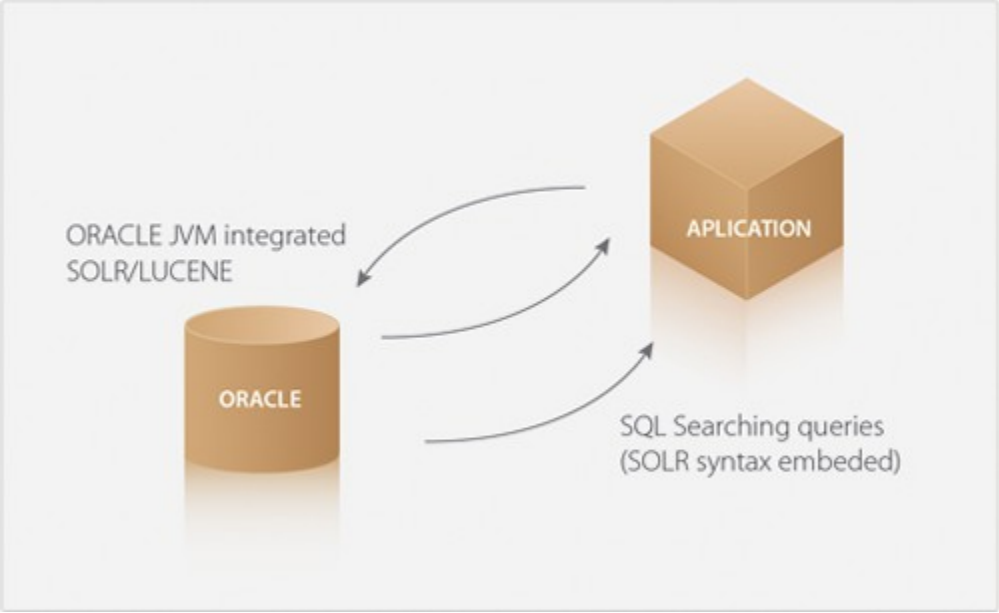
Apache Solr tutorial, a relational way

1. Introduction

When you have to perform searches over big data, you need specialized solutions that can deal with the velocity, variety and volume of this valuable information which analysis allows you to implement better solutions and to delineate appropriate business strategies. An important challenge in this area is how to handle the company's big data and normalized data in an efficient, on line and integrated way. Oracle® database has the ability to implement user's Java applications inside the engine in a transparently way.
Following this approach, Scotas OLS is a product that has all the previous characteristics, through the implementation of the market standard solution Solr/Lucene inside Oracle® engine adding transactional and security features.
Scotas OLS is specially designed for:

- Applications requiring certification against security and audits standards.
- Advanced Full-Text Search features.
- Applications requiring no delay between data changes and a positive hit.
- Near Real Time update/insert and Real Time deletes.
- On-Line index/rebuild, Parallel index/rebuild.



1.1 Technical overview

Scotas OLS is a tight integration of the popular, blazing fast open source enterprise search platform from the ***Apache Lucene*** project ***Solr*** with ***Oracle® 11g/12c*** Enterprise Edition.
Our implementation provides to the ***Oracle RDBMS*** the power of the ***Solr*** searching facilities such as faceting, geospatial and highlighting among others natively into the SQL.

Also by running ***Solr*** into the ***Oracle® RDBMS*** your SQL data is automatically indexed and updated in NRT (Near Real Time) way without any programming code.
For ***Solr*** users who want other features such as transactional storage, encryption, compression or row level filtering, for example, this integration is the perfect solution.

1.2 Architecture

OLS is tightly integrated into the ***Oracle® RDBMS*** by using the ***Oracle® Data Cartridge API*** (***ODCI***) adding a new Domain Index such as ***Oracle® Text*** or ***interMedia®*** -a domain index is like any other ***Oracle®*** index attached to some column of a specific table-.
This integration provides to ***Solr*** Engine the ability of running as another Oracle® server process receiving notifications when rows are added, changed or deleted. Also, it is a bi-directional integration provisioning to the SQL a new relational operator (***scontains***) and several new functions and procedures which can be used on SQL constructions or procedural code.

Following picture depicts the architecture of ***Scotas OLS***.

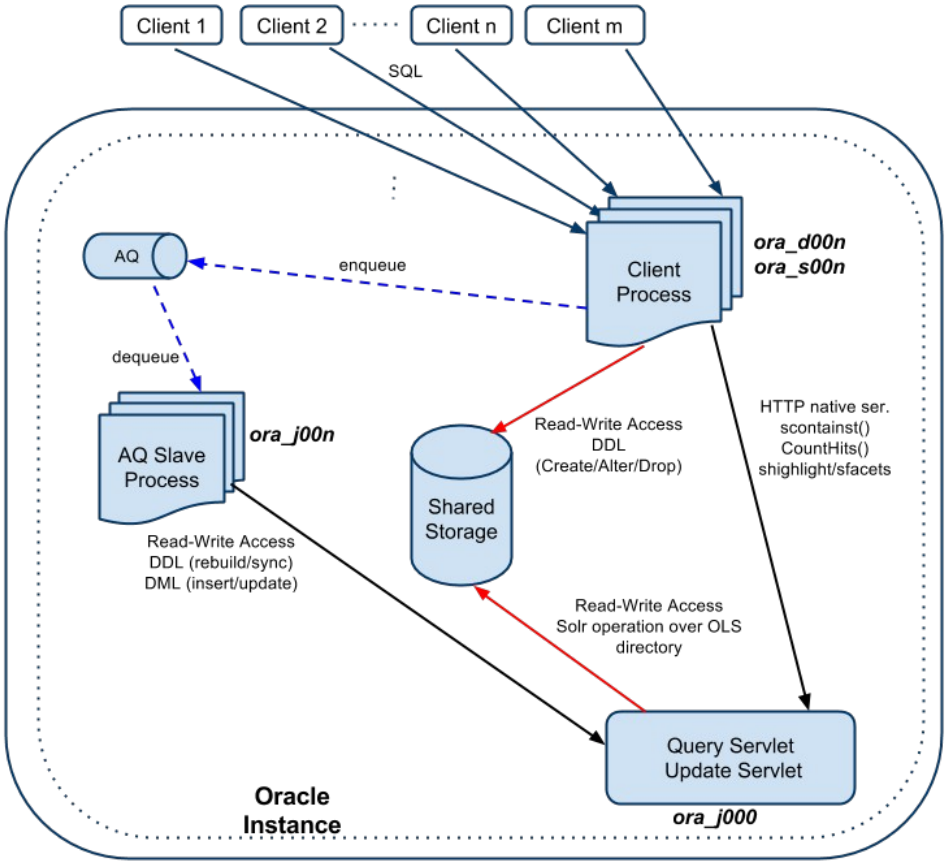


Figure 1: Diagram of Scotas OLS architecture.

Above picture shows how many processes interacts during ***OLS*** operation in an ***Oracle® RAC*** installation. External applications connects to the Oracle instance using transparent fail-over configuration (***TAF***) which routes the SQL sentences to some of the nodes of the installation.

Each connection to the RDBMS has associated an ***Oracle®*** process *ora_d00n* if is connected using a dedicated connection or an *ora_s00n* if is configured in shared mode.
Once a SQL query is interpreted by the engine several actions are started depending on if it is a DML operation or a query.
For DML operation once is committed the changes are propagated by the ***Oracle® AQ*** sub-system and the ***Solr*** instance running as separated server process receives the rowid(s) that was(were) changed updating their index structure, note that only ***rowid*** values are transmitted between processes, not the row data.
A parallel slave server process which is started during database startup runs a ***Solr*** server instance for one or all OLS index declared, the communication between this ***Solr*** instance and the client process which is associated to the client connection is using HTTP binary format. For DML operations ***Solr*** receives the ***rowid*** that need to be updated and if it is necessary gets the row values from the table(s) involved (these are processed using the internal OJVM drivers which have directly access to SGA areas of the RDBMS).
During SQL select operations the client associated process (***ora_d00n*** or ***ora_s00n***) talks to the ***Solr*** server instance using the start-fetch-close semantic, sending ***Solr*** query arguments, sorts, etc, during fetch stage, consuming ***rowid***(s) that match in the fetch stage and cleaning up any temporary structures at close.

1.3 Working modes

OLS works in two different modes depending on the kind of application's behavior.
For applications which have a high rate of DML operations by second on the table which the domain index is defined, a ***Deferred*** mode is better. In this scenario changes on ***Solr*** index are apply at an application specific point in time or periodically using the scheduler.
On the other hand, ***OnLine*** mode works in a *Near Real Time* mode (NRT), once the changes are committed they are applied into ***Solr*** index resulting in a positive hit if it was an insert, for example. In case of delete operations they are working in *Real Time* (RT) mode, similar to the SQL transaction behavior, once the row is deleted it will result in negative hit immediately for the current transaction and at commit point for others concurrent transactions.
The following trace diagram shows how the NRT/RT mode works in terms of data consistence in a concurrent multi-user environment.

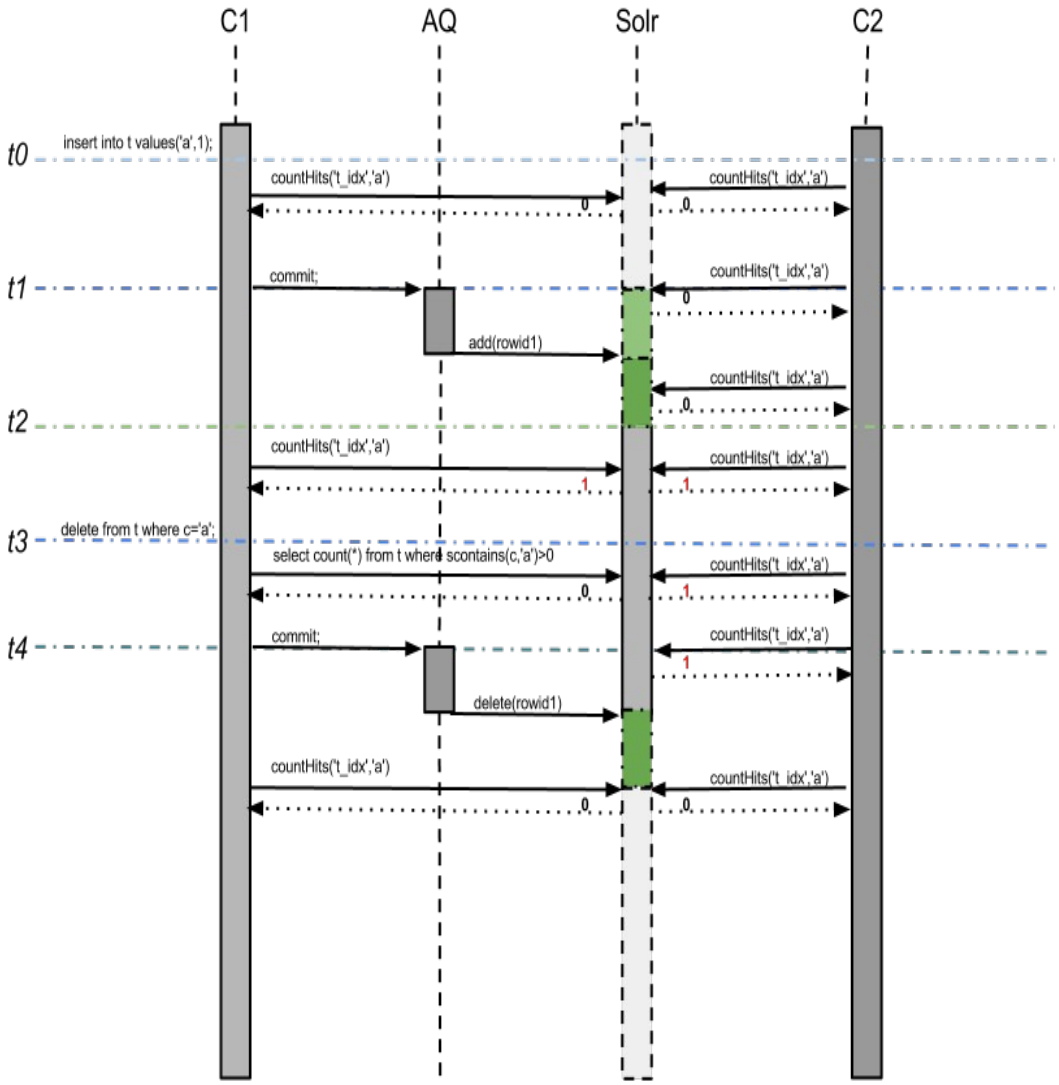


Figure 2: Diagram of NRT/RT mode work in terms of data consistence.

For DML insert (update) there is time frame t0-t2 where a negative hit is returned and differs from traditional SQL consistence, note that even if the C1 transaction inserts a row during this period (t0-t2) a call to ***countHits***() function results in a negative hit.

When C1 commits their changes these are processed by the AQ sub-system which sends a set of ***rowid***(s) for processing to ***Solr*** instance, during t1-t2 time frame (which is very small) also C2 connection gets a negative hit, this is why the mode is called *Near Real Time* to make a difference from SQL consistence mode, but the implementations results in high performance search sub-system for free text queries.

After t2 point in time the main transaction and others concurrent connections see a positive hit of the row just inserted.

Following the time-line a delete DML operation is shown, unlike inserts or updates, deletes can't be deferred and always works consistence in pair with traditional consistence level, this is because of the transaction C1 ***OLS*** domain index must no return a ***rowid*** which was deleted as

a positive hit (time frame t3-tn) but C2 connection is still sees the row as positive hit until C1 commit theirs changes (t3-t4), once C1 do a commit ***countHits***() function immediately returns a negative hit on C2 connection too.

1.4 Index storage

OLS replaces the ***Lucene*** inverted index storage, which by default is stored on the OS file-system, by ***Oracle® Secure File BLOBs***, resulting in high scalable, secure and transactional storage, see Secure Files whitepaper for more details and performance comparison against **NFS** or **ext3** file-system.
A summarizes advantages of this approach are:

- Transactional storage, a parallel process can do insert or optimize operations and if they fail simply do a rollback and nothing happens to other concurrent sessions.
- Compression and encryption using Secure File functionality, applicable to Lucene Inverted Index storage and ***Solr*** configuration files.
- Shared storage for ***Lucene*** Inverted Index, on RAC installations several processes across nodes can use the storage transparently.

2. Tutorial

This tutorial is relational version of Apache Solr Tutorial it is designed for Solr users which want to understand how Scotas OLS works.

2.1 Overview

This document covers the basics of running Scotas OLS using an example schema, and some sample data.

2.2 Requirements

- Scotas OLS/PushConnector installed
- SQLPlus to access to the RDBMS

2.3 Getting Started

Begin changing your working directory to be the "example" directory. (Note that the base directory name may vary with the version of Scotas OLS downloaded.) For example, with a shell in UNIX, Cygwin, or MacOS:

```
invitado@localhost:~/tmp/ols/db$ cd db/
invitado@localhost:~/tmp/ols/db$ cd tutorial/
invitado@localhost:~/tmp/ols/db/tutorial$ ll
total 52
drwxr-xr-x 3 invitado invitado  4096 jul 14 09:54 ./
drwxr-xr-x 3 invitado invitado  4096 jul 14 09:54 ../
drwxr-xr-x 3 invitado invitado  4096 jun 24 08:45 LUCENE.TUTORIAL_SIDX/
```

```
-rw-r--r-- 1 invitado invitado  8428 jul 14 09:54 schemaTutorial.sql
-rw-r--r-- 1 invitado invitado 26056 jul 14 09:54 testTutorial.xml
```

Content of directory LUCENE.TUTORIAL includes Solr configuration files which are installed by default into Scotas OLS.

```
invitado@localhost:~/tmp/ols/db/tutorial$ ll LUCENE.TUTORIAL_SIDX/conf/
total 136
drwxr-xr-x 2 invitado invitado  4096 jul 14 09:54 ./
drwxr-xr-x 3 invitado invitado  4096 jun 24 08:45 ../
-rw-r--r-- 1 invitado invitado   313 jul 14 09:54 elevate.xml
-rw-r--r-- 1 invitado invitado 78514 jul 14 09:54 mapping-FoldToASCII.txt
-rw-r--r-- 1 invitado invitado  2868 jul 14 09:54 mapping-ISOLatin1Accent.txt
-rw-r--r-- 1 invitado invitado    25 jul 14 09:54 protwords.txt
-rw-r--r-- 1 invitado invitado 15247 jul 14 09:54 schema.xml
-rw-r--r-- 1 invitado invitado  5326 jul 14 09:54 solrconfig.xml
-rw-r--r-- 1 invitado invitado    13 jul 14 09:54 spellings.txt
-rw-r--r-- 1 invitado invitado   134 jul 14 09:54 stopwords.txt
-rw-r--r-- 1 invitado invitado   474 jul 14 09:54 synonyms.txt
```

Before starting with this tutorial be sure that Scotas OLS is installed and running and SCOTT have the role for using it.

```
invitado@localhost:~/tmp/ols/db/tutorial$ sqlplus scott/tiger@test_ols
SQL*Plus: Release 12.1.0.2.0 Production on Fri Oct 17 11:08:40 2014

Copyright (c) 1982, 2014, Oracle.  All rights reserved.
Last Successful login time: Fri Oct 17 2014 09:42:15 -03:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options
SQL> select * from session_roles;
ROLE
--------------------------------------------------------------------------------
CONNECT
RESOURCE
LUCENEUSER

 SQL>  select bg_process_name,port from lucene.bg_process;
```

BG_PROCESS_NAME	PORT
IndexScanServ	1099
IndexUpdateServ	1098
SolrServlet	9099
SolrServlet	9199

You can change OLS logging by using Logging Servlet on each ports (http://localhost:9099/logging/, http://localhost:9199/logging/).

In case you are using Scotas Push Connector check if external Solr server is running in the ***hostname*** and ***port*** described by ***bg_process*** table, for example visiting the URL http://localhost:8983/solr/

2.4 Indexing Data

2.4.1 Solr way

Your Solr server is up and running, but it doesn't contain any data. You can modify a Solr index by POSTing commands to Solr to add (or update) documents, delete documents, and commit pending adds and deletes. These commands can be in a variety of formats.

The ***exampledocs*** directory contains sample files showing of the types of commands Solr accepts, as well as a java utility for posting them from the command line (a post.sh shell script is also available, but for this tutorial we'll use the cross-platform Java client. Run ***java -jar post.jar -h*** so see it's various options).

To try this, open a new terminal window, enter the ***exampledocs*** directory, and run "***java -jar post.jar***" on some of the XML files in that directory.

```
user:~/solr/example/exampledocs$ java -jar post.jar solr.xml monitor.xml
SimplePostTool: version 1.4
SimplePostTool: POSTing files to http://localhost:8983/solr/update..
SimplePostTool: POSTing file solr.xml
SimplePostTool: POSTing file monitor.xml
SimplePostTool: COMMITting Solr index changes..
```

You have now indexed two documents in Solr, and committed these changes. You can now search for "solr" by loading the "Query" tab in the Admin interface, and entering "solr" in the "q" text box. Clicking the "Execute Query" button should display the following URL containing one result...

http://localhost:8983/solr/collection1/select?q=solr&wt=xml

2.4.2 Relational way

The db/tutorial directory have a file named ***schemaTutorial.sql*** lets execute this step by step.

```
invitado@localhost:~/tmp/ols/db/tutorial$ sqlplus scott/tiger@test_ols
SQL*Plus: Release 12.1.0.2.0 Production on Fri Oct 17 11:43:18 2014

Copyright (c) 1982, 2014, Oracle.  All rights reserved.

Last Successful login time: Fri Oct 17 2014 11:37:49 -03:00

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.2.0 - 64bit Production
With the Partitioning, OLAP, Advanced Analytics and Real Application Testing options

SQL> CREATE TYPE CAT_LIST_ARR AS VARRAY(20) OF VARCHAR2(100) ;
/
Type created.

SQL> CREATE TABLE OLS_TUTORIAL (
 ID VARCHAR2(30) PRIMARY KEY,
 NAME VARCHAR2(400),
 MANU VARCHAR2(4000),
 CAT  cat_list_arr,
```

```
 FEATURES CLOB,
 INCLUDES VARCHAR2(4000),
 WEIGHT   NUMBER,
 PRICE    NUMBER,
 POPULARITY NUMBER,
 INSTOCK    CHAR(5), -- true or false
 MANUFACTUREDATE_DT TIMESTAMP,
 PAYLOADS   VARCHAR2(4000),
 STORE   VARCHAR2(200))
;
Table created.

SQL> set define off
SQL> insert into ols_tutorial values ( 'SOLR1000',
'Solr, the Enterprise Search Server',
'Apache Software Foundation',
CAT_LIST_ARR('software','search'),
'Advanced Full-Text Search Capabilities using Lucene
Optimized for High Volume Web Traffic
Standards Based Open Interfaces - XML and HTTP
Comprehensive HTML Administration Interfaces
Scalability - Efficient Replication to other Solr Search Servers
Flexible and Adaptable with XML configuration and Schema
Good unicode support: héllo (hello with an accent over the e)',
NULL,
NULL,
0,
10,
'true',
TO_TIME  2  STAMP('2006-01-17T00:00:00Z','YYYY-MM-DD"T"HH24:MI:SS"Z"'),
NULL,
NULL)
;
1 row created.
SQL> insert into ols_tutorial values ( '3007WFP',
'Dell Widescreen UltraSharp 3007WFP',
'Dell, Inc.',
CAT_LIST_ARR('electronics','monitor'),
'30" TFT active matrix LCD, 2560 x 1600, .25mm dot pitch, 700:1 contrast',
'USB cable',
401.6,
2199,
6,
'true',
NULL,
NULL,
'43.17614,-90.57341')
;
1 row created.
SQL> commit;
Commit complete.
SQL> select count(*) from ols_tutorial;
  COUNT(*)
  ----------
           2
-- indexing
SQL> CREATE INDEX TUTORIAL_SIDX ON OLS_TUTORIAL(ID) INDEXTYPE IS LUCENE.SOLRINDEX
parameters('Searcher:0;Updater:0;NormalizeScore:true;SyncMode:OnLine;CommitOnSync:true;SoftCommit:true;LockMast
```

erTable:false;IncludeMasterColumn:false;LogLevel:ALL;MltColumn:name_tg;HighlightColumn:name_tg,features;FacetedCols:cat_tw,price_f,inStock_b,manufacturedate_dt;DefaultColumn:text;ExtraCols:*id "id_s",cat "cat_tw",name "name_tg",features "features",manu "manu_tg",includes "includes_t",price "price_f",popularity "popularity_i",inStock "inStock_b",manufacturedate_dt "manufacturedate_dt",store "location_pn"*');
Index created.
SQL> select id,name from ***ols_tutorial*** where **scontains*(id,'solr')>0***;

ID	NAME
SOLR1000	Solr, the Enterprise Search Server

2.4.3 Adding more documents Solr way

You can index all of the sample data, using the following command (assuming your command line shell supports the *.xml notation):

```
user:~/solr/example/exampledocs$ java -jar post.jar *.xml
SimplePostTool: version 1.4
SimplePostTool: POSTing files to http://localhost:8983/solr/update..
SimplePostTool: POSTing file gb18030-example.xml
SimplePostTool: POSTing file hd.xml
SimplePostTool: POSTing file ipod_other.xml
SimplePostTool: POSTing file ipod_video.xml
...
SimplePostTool: POSTing file solr.xml
SimplePostTool: POSTing file utf8-example.xml
SimplePostTool: POSTing file vidcard.xml
SimplePostTool: COMMITting Solr index changes..
```

2.4.4 Adding more documents Relational way

An this step the index and table is already created, adding more rows is simple and once we commit rows Scotas OLS index is updated.

```
SQL> insert into ols_tutorial values ( 'SP2514N',
'Samsung SpinPoint P120 SP2514N - hard drive - 250 GB – ATA-133',
'Samsung Electronics Co. Ltd.',
CAT_LIST_ARR('electronics','hard-drive'),
'7200RPM, 8MB cache, IDE Ultra ATA-133
NoiseGuard, SilentSeek technology, Fluid Dynamic Bearing (FDB) motor',
NULL,
NULL,
92,
6,
'true',
TO_TIMESTAMP('2006-02-13T15:26:37Z','YYYY-MM-DD"T"HH24:MI:SS"Z"'),
NULL,
'35.0752,-97.032')
;
-- Near Oklahoma city
….. rest of the other 14 docs.....
insert into ols_tutorial values ( '100-435805',
'ATI Radeon X1900 XTX 512 MB PCIE Video Card',
'ATI Technologies',
CAT_LIST_ARR('electronics','graphics-card'),
'ATI RADEON X1900 GPU/VPU clocked at 650MHz
512MB GDDR3 SDRAM clocked at 1.55GHz
```

```
PCI Express x16
dual DVI, HDTV, svideo, composite out
OpenGL 2.0, DirectX 9.0',
NULL,
48,
649.99,
7,
'false',
TO_TIMESTAMP('2006-02-13T15:26:37Z','YYYY-MM-DD"T"HH24:MI:SS"Z"'),
NULL,
'40.7143,-74.006')
;
-- NYC store
SQL> commit;
Commit complete.
```

2.4.5 Simple search Solr way

You can search for all sorts of things using the default Solr Query Syntax (a superset of the Lucene query syntax)...

video

name:video

+video +price:[* TO 400]

There are many other different ways to import your data into Solr... one can

- Import records from a database using the Data Import Handler (DIH).
- Load a CSV file (comma separated values), including those exported by Excel or MySQL.
- POST JSON documents
- Index binary documents such as Word and PDF with Solr Cell (ExtractingRequestHandler).
- Use SolrJ for Java or other Solr clients to programatically create documents to send to Solr.

2.4.6 Simple search Relational way

Above Solr search links are equivalent in SQL as:

```
SQL> select sscore(1) sc,id,name from ols_tutorial where scontains(id,'video',1)>0;
```

SC	ID	NAME
1	100-435805	ATI Radeon X1900 XTX 512 MB PCIE Video Card
.875	MA147LL/A	Apple 60 GB iPod with Video Playback Black

```
SQL> select sscore(1) sc,id,name from ols_tutorial where scontains(id,'name_tg:video',1)>0;
```

SC	ID	NAME
1	100-435805	ATI Radeon X1900 XTX 512 MB PCIE Video Card

SC	ID	NAME
.833333433	MA147LL/A	Apple 60 GB iPod with Video Playback Black

SQL> select id,name,price from **ols_tutorial** where **scontains(id,'+video +price_f:[* TO 400]',1)>0**;

ID	NAME	PRICE
MA147LL/A	Apple 60 GB iPod with Video Playback Black	399

Note that unlike in Solr example field column name was indexed as ***name_tg*** and price as ***price_f***, this was intentional to consider this field as *Text General* (name) and *Float* (price).

2.5 Updating Data

You may have noticed that even though the file ***solr.xml*** has now been POSTed to the server twice, you still only get 1 result when searching for "solr". This is because the example ***schema.xml*** specifies a "***uniqueKey***" field called "***id***". Whenever you POST commands to Solr to add a document with the same value for the ***uniqueKey*** as an existing document, it automatically replaces it for you. You can see that that has happened by looking at the values for ***numDocs*** and ***maxDoc*** in the "CORE"/searcher section of the statistics page...

http://localhost:8983/solr/#/collection1/plugins/core?entry=searcher

numDocs represents the number of searchable documents in the index (and will be larger than the number of XML files since some files contained more than one <doc>). ***maxDoc*** may be larger as the ***maxDoc*** count includes logically deleted documents that have not yet been removed from the index. You can re-post the sample XML files over and over again as much as you want and ***numDocs*** will never increase, because the new documents will constantly be replacing the old.

Go ahead and edit the existing XML files to change some of the data, and re-run the ***java -jar post.jar*** command, you'll see your changes reflected in subsequent searches.

2.5.1 Updating data relational way

Unlike Solr, default ***schema.xml*** file used by OLS defines ***rowid*** as ***uniqueKey*** this because an Oracle ***rowid*** is the unique way to identified any row in the database world, here sample section of default OLS ***schema.xml***:

```
<fields>
  <field name="rowid" type="string" indexed="true" stored="true" required="true" />
....
 </fields>
 <uniqueKey>rowid</uniqueKey>
..
```

Apart from that updating is just updating and committing changes, for example:

```
SQL> SELECT ID,NAME,PRICE FROM OLS_TUTORIAL WHERE SCONTAINS(ID,'name_tg:"Server-mod"')>0;
```

ID	NAME	PRICE
no rows selected		

```
SQL> UPDATE OLS_TUTORIAL SET id=id,NAME = 'Solr, the Enterprise Search Server-mod' WHERE ID='SOLR1000';
1 row updated.
SQL> COMMIT;
```

Commit complete.
SQL> SELECT ID,NAME,PRICE FROM OLS_TUTORIAL WHERE SCONTAINS(ID,'***name_tg:"Server-mod***"')>0;

ID	NAME	PRICE
SOLR1000	Solr, the Enterprise Search Server-mod	0

Note that is described into Figure 2 at the beginning of this document OLS is working in Near Real Time mode which means that after commit there is a time (milliseconds) between the change and a positive hits returned by ***SCONTAINS*** operator.

2.6 Deleting Data

You can delete data by POSTing a delete command to the update URL and specifying the value of the document's unique key field, or a query that matches multiple documents (be careful with that one!). Since these commands are smaller, we will specify them right on the command line rather than reference an XML file.

2.6.1 Deleting data Solr way

Execute the following command to delete a specific document

```
java -Ddata=args -Dcommit=false -jar post.jar "<delete><id>SP2514N</id></delete>"
```

Because we have specified "commit=false", a search for id:SP2514N we still find the document we have deleted. Since the example configuration uses Solr's "***autoCommit***" feature Solr will still automatically persist this change to the index, but it will not affect search results until an "***openSearcher***" commit is explicitly executed.

Using the statistics page for the ***updateHandler*** you can observe this delete propogate to disk by watching the ***deletesById*** value drop to 0 as the ***cumulative_deletesById*** and ***autocommit*** values increase.

Here is an example of using delete-by-query to delete anything with DDR in the name:

```
java -Dcommit=false -Ddata=args -jar post.jar "<delete><query>name:DDR</query></delete>"
```

You can force a new searcher to be opened to reflect these changes by sending an explicit commit command to Solr:

```
java -jar post.jar -
```

Now re-execute the previous search and verify that no matching documents are found. You can also revisit the statistics page and observe the changes to both the number of commits in the updateHandler and the ***numDocs*** in the searcher.

Commits that open a new searcher can be expensive operations so it's best to make many changes to an index in a batch and then send the commit command at the end. There is also an optimize command that does the same things as commit, but also forces all index segments to be merged into a single segment -- this can be very resource intensive, but may be worthwhile for improving search speed if your index changes very infrequently.

All of the update commands can be specified using either XML or JSON.

To continue with the tutorial, re-add any documents you may have deleted by going to the ***exampledocs*** directory and executing

```
java -jar post.jar *.xml
```

2.6.2 Deleting data relational way

As in the updating data section deleting data is just using SQL delete DML operation. Here an example deleting row by id:

```
SQL> SELECT ID,NAME,PRICE FROM OLS_TUTORIAL WHERE SCONTAINS(ID,'id_s:SP2514N')>0;
```

ID	NAME	PRICE
SP2514N	Samsung SpinPoint P120 SP2514N - hard drive - 250 GB - ATA-133	0

```
SQL> DELETE FROM OLS_TUTORIAL WHERE ID='SP2514N';
1 row deleted.
SQL> COMMIT;
Commit complete.
SQL> SELECT ID,NAME,PRICE FROM OLS_TUTORIAL WHERE SCONTAINS(ID,'id_s:SP2514N')>0;
```

ID	NAME	PRICE
no rows selected		

Here an example deleting by query:

```
SQL> SELECT ID,NAME,PRICE FROM OLS_TUTORIAL WHERE SCONTAINS(ID,'name_tg:DDR')>0;
```

ID	NAME	PRICE
VS1GB400C3	CORSAIR ValueSelect 1GB 184-Pin DDR SDRAM Unbuffered DDR 400 (PC 3200) System Me mory - Retail	74.99
VDBDB1A16	A-DATA V-Series 1GB 184-Pin DDR SDRAM Unbuffered DDR 400 (PC 3200) System Memory	
TWINX2048-3200PRO	CORSAIR XMS 2GB (2 x 1GB) 184-Pin DDR SDRAM Unbuffered DDR 400 (PC 3200) Dual C hannel Kit System Memory - Retail	185

```
SQL> DELETE FROM OLS_TUTORIAL WHERE SCONTAINS(id,'name_tg:DDR')>0;
3 rows deleted.
SQL> COMMIT;
Commit complete.
SQL> SELECT ID,NAME,PRICE FROM OLS_TUTORIAL WHERE SCONTAINS(ID,'name_tg:DDR')>0;
```

ID	NAME	PRICE
no rows selected		

Note that unlike update ***DML*** command which works in ***NRT*** mode, delete ***DML*** command is always working in Real Time mode which means that even if Solr index is not yet updated no phantom rows are returned by ***SCONTAINS*** operator, see figure 2 above.

2.7 Querying Data

Searches are done via HTTP GET on the select URL with the query string in the q parameter.

2.7.1 Querying data Solr way

You can pass a number of optional request parameters to the request handler to control what information is returned. For example, you can use the "fl" parameter to control what stored fields are returned, and if the relevancy score is returned:

- q=video&fl=name,id (return only name and id fields)
- q=video&fl=name,id,score (return relevancy score as well)
- q=video&fl=*,score (return all stored fields, as well as relevancy score)
- q=video&sort=price desc&fl=name,id,price (add sort specification: sort by price descending)
- q=video&wt=json (return response in JSON format)

The query form provided in the web admin interface allows setting various request parameters and is useful when testing or debugging queries.

2.7.1.1 Sorting

Solr provides a simple method to sort on one or more indexed fields. Use the "sort' parameter to specify "field direction" pairs, separated by commas if there's more than one sort field:

- q=video&sort=price desc
- q=video&sort=price asc
- q=video&sort=inStock asc, price desc

"***score***" can also be used as a field name when specifying a sort:

- q=video&sort=score desc
- q=video&sort=inStock asc, score desc

Complex functions may also be used to sort results:

- q=video&sort=div(popularity,add(price,1)) desc

If no sort is specified, the default is ***score desc*** to return the matches having the highest relevancy.

2.7.2 Querying relational way

We see some example of querying data in previous examples, lets do above Solr queries using relational way:

```
SQL> SELECT ID,NAME FROM OLS_TUTORIAL T WHERE SCONTAINS(ID,'video')>0;
```

ID	NAME
100-435805	ATI Radeon X1900 XTX 512 MB PCIE Video Card
MA147LL/A	Apple 60 GB iPod with Video Playback Black

```
SQL> SELECT sscore(1) SC,ID,NAME FROM OLS_TUTORIAL T WHERE SCONTAINS(ID,'video',1)>0;
```

SC	ID	NAME
1	100-435805	ATI Radeon X1900 XTX 512 MB PCIE Video Card
0.875	MA147LL/A	Apple 60 GB iPod with Video Playback Black

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ ID,PRICE,NAME FROM OLS_TUTORIAL T
        WHERE SCONTAINS(ID,'video','price_f desc')>0;
```

ID	PRICE	NAME
100-435805	649.99	ATI Radeon X1900 XTX 512 MB PCIE Video Card
MA147LL/A	399	Apple 60 GB iPod with Video Playback Black

Note that for using ***SSCORE()*** ancillary operator there is correlation parameter to SCONTAINS() that must match to work, in above example ***SCORE(1)*** matches with ***SCONTAINS(..,..,1)***>0.

There are four overloaded version of ***SCONTAINS***:

1. SCONTAINS(col,'qry')
2. SCONTAINS(col,'qry',correlation_id)
3. SCONTAINS(col,'qry','sort_string')
4. SCONTAINS(col,'qry','sort_string',correlation_id)

format 1 and 4 can be used in conjunction with ***SSCORE(correlation_id)***.

Unlike Solr there is no necessary to use ***fl*** parameter to define which Solr field must be returned by a query, ***SCONSTAIN*** is part of a regular SQL query so which column is returned is defined in select section of the SQL syntax.

2.7.2.1 Sorting

Here another examples with domain index level sorting:

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ ID,PRICE,NAME FROM OLS_TUTORIAL T
        WHERE SCONTAINS(ID,'video','price_f desc')>0;
```

ID	PRICE	NAME
100-435805	649.99	ATI Radeon X1900 XTX 512 MB PCIE Video Card
MA147LL/A	399	Apple 60 GB iPod with Video Playback Black

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ ID,PRICE,NAME FROM OLS_TUTORIAL T
        WHERE SCONTAINS(ID,'video','price_f asc')>0;
```

ID	PRICE	NAME
MA147LL/A	399	Apple 60 GB iPod with Video Playback Black
100-435805	649.99	ATI Radeon X1900 XTX 512 MB PCIE Video Card

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ ID,PRICE,INSTOCK FROM OLS_TUTORIAL T
        WHERE SCONTAINS(ID,'video','inStock_b asc,price_f desc')>0
```

ID	PRICE	INSTOCK
MA147LL/A	649.99	false
100-435805	399	true

Sort string is comma separated list of ***field_name [asc|desc]*** syntax, note that we are using ***field_name*** not column name because during index creation time we named OLS/Solr field names different from column name mainly to use ***schema.xml*** pattern definitions for fields. For example column name ***PRICE*** where defined as field name ***price_f***, default ***schema.xml*** defines fields ending with ****_f*** as Solr type float.

Is important remember to add ***DOMAIN_INDEX_SORT*** optimizer hint to prevent ***RDBMS*** to return rows in other order rather than the domain index (for example using physical block order).

"***score***" can also be used as a field name when specifying a sort:

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ round(sscore(1),2) sc,ID,NAME FROM OLS_TUTORIAL T
     WHERE SCONTAINS(ID,'video','score desc',1)>0;
```

SC	ID	NAME
1	100-435805	ATI Radeon X1900 XTX 512 MB PCIE Video Card
0.88	MA147LL/A	Apple 60 GB iPod with Video Playback Black

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ round(sscore(1),2) sc,ID,INSTOCK,NAME FROM OLS_TUTORIAL T
     WHERE SCONTAINS(ID,'video','inStock_b asc,score desc',1)>0;
```

SC	ID	INSTOCK	NAME
1	100-435805	false	ATI Radeon X1900 XTX 512 MB PCIE Video Card
0.88	MA147LL/A	true	Apple 60 GB iPod with Video Playback Black

By default ***score desc*** is the natural order returned by ***SCONTAINS*** operator.

Complex functions may also be used to sort results, for example math or ***geo-localization***:

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ round(POPULARITY/(PRICE+1),2) PP,NAME FROM OLS_TUTORIAL T
     WHERE SCONTAINS(ID,'video','div(popularity_i,add(price_f,1)) desc',1)>0;
```

PP	NAME
0.03	ATI Radeon X1900 XTX 512 MB PCIE Video Card
0.01	Apple 60 GB iPod with Video Playback Black

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ id,round(SSCORE(1),2) sc,name FROM OLS_TUTORIAL T
     where scontains(id,'video','geodist(location_pn,0.0,0.0) desc',1)>0;
```

ID	SC	NAME
MA147LL/A	0.88	Apple 60 GB iPod with Video Playback Black
100-435805	1	ATI Radeon X1900 XTX 512 MB PCIE Video Card

If no sort string is specified in ***SCONTAINS***, the default is ***score desc*** to return the matches having the highest relevancy.

2.8 Highlighting

Hit highlighting returns relevant snippets of each returned document, and highlights terms from the query within those context snippets.

2.8.1 Solr way

The following example searches for video card and requests highlighting on the fields name,features. This causes a highlighting section to be added to the response with the words to highlight surrounded with ****** (for emphasis) tags.

...&q=video card&fl=name,id&hl=true&hl.fl=name,features

More request parameters related to controlling highlighting may be found here.

2.8.2 Relational way

Highlighting is supported by ancillary operator ***shighlight()*** for example:

SQL> SELECT ***shighlight***(1) hl,id FROM OLS_TUTORIAL T WHERE SCONTAINS(ID,'features(video card)',1)>0;

HL	ID
Hi-Speed USBmemory card: CompactFlash, Micro Drive, SmartMedia, Memory Stick, Memory Stick Pro, SD Card, and MultiMediaCard	0579B002
Apple 60 GB iPod with Video Playback Black iTunes, Podcasts, Audiobooks Stores up to 15,000 songs, 25,000 photos, or 150 hours of video2.5-inch	MA147LL/A
16Dual DVI connectors, HDTV out, video inputOpenGL 2.0, DirectX 9.0	EN7800GTX/2DHTV/256M

shighlight() operator works with fields defined during index creation time as parameter ***HighlightColumn***, for example ***HighlightColumn:name_tg,features***.

This parameter could be changed at any time using alter index parameter DDL command.

2.9 Faceted Search

Faceted search takes the documents matched by a query and generates counts for various properties or categories. Links are usually provided that allows users to "drill down" or refine their search results based on the returned categories.

2.9.1 Solr

The following example searches for all documents (*:*) and requests counts by the category field cat.

...&q=*:*&facet=true&facet.field=cat

Notice that although only the first 10 documents are returned in the results list, the facet counts generated are for the complete set of documents that match the query.

We can facet multiple ways at the same time. The following example adds a facet on the boolean ***inStock*** field:

...&q=*:*&facet=true&facet.field=cat&facet.field=inStock

Solr can also generate counts for arbitrary queries. The following example queries for ipod and shows prices below and above 100 by using range queries on the price field.

...&q=ipod&facet=true&facet.query=price:[0 TO 100]&facet.query=price:[100 TO *]

Solr can even facet by numeric ranges (including dates). This example requests counts for the manufacture date (***manufacturedate_dt*** field) for each year between 2004 and 2010.

...&q=*:*&facet=true&facet.range=manufacturedate_dt&facet.range.start=2004-01-01T00:00:00Z&facet.range.end=2010-01-01T00:00:00Z&facet.range.gap=+1YEAR

More information on faceted search may be found on the faceting overview and faceting parameters pages.

2.9.2 Relational

Faceting functionality is implemented by a pipe-line table function named ***SFACETS***, a pipe-line table function returns a set of rows which can be evaluated using ***TABLE*** operator.

```
SQL> select * from TABLE(SELECT T.FACETS F FROM
        TABLE(SFACETS(USER||'.TUTORIAL_SIDX','*:*','facet.field=cat_tw')) T);
```

NAME	VALUE
electronics	10
connector	2
graphics-card	2
monitor	2
search	2
software	2
camera	1
copier	1
hard-drive	1
multifunction-printer	1
music	1
printer	1
scanner	1
memory	0

First argument of ***SFACETS*** pipe-line table function is the index name using syntax ***[schema.]index_name***, second parameter is in Solr query syntax as in ***q*** parameter of above examples, last argument is any other faceting parameters using the URL syntax ***name=value***.

```
SQL> select * from table(SELECT T.FACETS F FROM
        TABLE(SFACETS(USER||'.TUTORIAL_SIDX','name_tg:video','facet.field=cat_tw&facet.mincount=1')) T);
```

NAME	VALUE
electronics	2
graphics-card	1
music	1

```
SQL> SELECT FIELD,SJOIN(T.FACETS) F FROM
        TABLE(SFACETS(USER||'.TUTORIAL_SIDX','*:*','facet.field=cat_tw&facet.field=inStock_b')) T;
```

FIELD	F
cat_tw	electronics(10),connector(2),graphics-card(2),monitor(2),search(2),software(2),camera(1),copier(1),hard-drive(1),multifunction-printer(1),music(1),printer(1),scanner(1),memory(0)
inStock_b	true(8),false(4)

SJOIN function converts a ***VARRAY*** type to a comma concatenated string of ***name(value)*** format.

```
SQL> select * from table(SELECT T.QUERIES FROM
        TABLE(SFACETS(USER||'.TUTORIAL_SIDX','*:*',
            'facet.query=price_f:[0+TO+100]&facet.query=price_f:[100+TO+*]')) T);
```

NAME	VALUE
price_f:[0 TO 100]	4
price_f:[100 TO *]	8

```
SQL> select * from table(SELECT T.DATES FROM
        TABLE(SFACETS(USER||'.TUTORIAL_SIDX','*:*',
        'facet.date=manufacturedate_dt&facet.date.start=2004-01-01T00:00:00Z&facet.date.end=2010-01-01T00:00:00Z&facet.date.gap=%2B1YEAR')) T)
```

NAME	VALUE
2004-01-01T00:00:00Z	0
2005-01-01T00:00:00Z	2
2006-01-01T00:00:00Z	6
2007-01-01T00:00:00Z	0
2008-01-01T00:00:00Z	0
2009-01-01T00:00:00Z	0
gap:+1YEAR	0
start:Wed Dec 31 21:00:00 ART 2003	0
end:Thu Dec 31 21:00:00 ART 2009	0

Last argument is automatically encoded using URL syntax except for the sign **+** which must be encoded as ***%2B***.

```
SQL> SELECT FIELD,SJOIN(T.FACETS) F,SJOIN(T.DATES) D FROM
        TABLE(SFACETS(USER||'.TUTORIAL_SIDX','*:*','
        facet.field=cat_tw&facet.field=inStock_b&facet.date=manufacturedate_dt&facet.date.start=2004-01-01T00:00:00Z&facet.date.end=2010-01-01T00:00:00Z&facet.date.gap=%2B1YEAR')) T
```

FIELD	F	D
cat_tw,manufacturedate_dt	Electronics(10), connector(2), graphics-card(2), monitor(2), search(2), software(2),	2004-01-01T00:00:00Z(0), 2005-01-01T00:00:00Z(2), 2006-01-01T00:00:00Z(6), 2007-01-01T00:00:00Z(0), 2008-01-01T00:00:00Z(0), 2009-01-01T00:00:00Z(0),

FIELD	F	D
	camera(1), copier(1), hard-drive(1), multifunction-printer(1), music(1),printer(1),scanner(1),memory(0)electronics(10),connector(2),graphics-card(2),monitor(2),search(2),software(2),camera(1),copier(1),hard-drive(1),multifunction-printer(1),music(1),printer(1),scanner(1),memory(0)	gap:+1YEAR(0), start:Wed Dec 31 21:00:00 ART 2003(0), end:Thu Dec 31 21:00:00 ART 2009(0)
inStock_b	true(8),false(4)	

Remember that ***SQLPlus*** or ***SQLDeveloper*** interprets ***&*** sign as escape character, so to execute above facet examples first execute "***set define off***".

2.10 Text Analysis

Text fields are typically indexed by breaking the text into words and applying various transformations such as lowercasing, removing plurals, or stemming to increase relevancy. The same text transformations are normally applied to any queries in order to match what is indexed.

The schema defines the fields in the index and what type of analysis is applied to them. The current schema your collection is using may be viewed directly via the Schema tab in the Admin UI, or explored dynamically using the Schema Browser tab.

The best analysis components (tokenization and filtering) for your textual content depends heavily on language. As you can see in the Schema Browser, many of the fields in the example schema are using a fieldType named text_general, which has defaults appropriate for most languages.

If you know your textual content is English, as is the case for the example documents in this tutorial, and you'd like to apply English-specific stemming and stop word removal, as well as split compound words, you can use the text_en_splitting fieldType instead. Go ahead and edit the schema.xml in the solr/example/solr/collection1/conf directory, to use the text_en_splitting fieldType for the text and features fields like so:

```
<field name="features" type="text_en_splitting" indexed="true" stored="true" multiValued="true"/>
...
<field name="text" type="text_en_splitting" indexed="true" stored="false" multiValued="true"/>
```

2.10.1 Solr

Stop and restart Solr after making these changes and then re-post all of the example documents using ***java -jar post.jar *.xml***. Now queries like the ones listed below will demonstrate English-specific transformations:

- A search for power-shot can match ***PowerShot***, and adata can match ***A-DATA*** by using the ***WordDelimiterFilter*** and ***LowerCaseFilter***.
- A search for features:recharging can match ***Rechargeable*** using the stemming features of ***PorterStemFilter***.
- A search for "1 gigabyte" can match ***1GB***, and the commonly misspelled pixima can matches ***Pixma*** using the ***SynonymFilter***.

A full description of the analysis components, ***Analyzers***, ***Tokenizers***, and ***TokenFilters*** available for use is here.

2.10.2 Relational

Using query syntax we reproduce above text analysis examples as:

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ SSCORE(1),name FROM OLS_TUTORIAL T
        WHERE SCONTAINS(ID,'power-shot',1)>0;
```

SSCORE(1)	NAME
1	Canon ***PowerShot*** SD500

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ SSCORE(1),NAME,features FROM OLS_TUTORIAL T
        WHERE SCONTAINS(ID,'features:recharging',1)>0;
```

SSCORE(1)	NAME	FEATURES
1	Apple 60 GB iPod with Video Playback Black	"iTunes, Podcasts, Audiobooks Stores up to 15,000 songs, 25,000 photos, or 150 hours of video 2.5-inch, 320x240 color TFT LCD display with LED backlight Up to 20 hours of battery life Plays AAC, MP3, WAV, AIFF, Audible, Apple Lossless, H.264 video Notes, Calendar, Phone book, Hold button, Date display, Photo wallet, Built-in games, JPEG photo playback, Upgradeable firmware, USB 2.0 compatibility, Playback speed control, ***Rechargeable*** capability, Battery level indication"

```
SQL> SELECT /*+ DOMAIN_INDEX_SORT */ SSCORE(1),NAME FROM OLS_TUTORIAL T
        WHERE SCONTAINS(ID,'pixima',1)>0;
```

SSCORE(1)	NAME
1	Canon ***PIXMA*** MP500 All-In-One Photo Printer

2.11 More Like this

Although Solr tutorial do not includes more like this example OLS tutorial have some simple examples about that feature:

```
SQL> select id,name from ols_tutorial where rowid in
        (select column_value from table(SELECT smlt(1)
        FROM OLS_TUTORIAL T WHERE SCONTAINS(ID,'pixima',1)>0 and rownum=1));
```

ID	NAME
9885A004	***Canon*** PowerShot SD500

As in highlight example there is an ancillary operator ***SMLT()*** which returns a list of ***rowids*** for rows which are similar to the current matching row, above text analysis example shows that if you are looking for ***pixima***, a row having name as "***Canon*** PIXMA MP500 All-In-One Photo Printer" match, so a similar product to it is " ***Canon*** PowerShot SD500".

More like this operator works using an index level parameter named ***MltColumn***, above index creation commands defines it as ***MltColumn:name_tg***, this parameter could be changed at any time.

3 Conclusion

Congratulations! You successfully ran a small OLS test, added some documents, and made changes to the index. You learned about queries and text analysis. You're ready to start using OLS on your own project!

OLS has a ton of other features that we haven't touched on here, including parallel/partitioning search to handle huge document collections, function queries, numeric field statistics, and son on.