Oracle Open World Data-and-Compute-Intensive processing Use Case: Lucene Domain Index

Marcelo F. Ochoa

Fac. Cs. Exactas - UNICEN - Tandil - Argentina

Agenda

- Data-and-Compute Intensive Search
- What is Lucene?
- What is Lucene Domain Index?
- Performance
- Application integration
- Demo
- Future plans

Data-and-Compute Intensive Search

- Data-and-Compute Intensive Search
  - Strategies
    - Middle-tier-based search engines
      - Google Appliance
      - SES
      - Nutch
      - Solr
    - Database-embedded search engines
      - Oracle Text
      - Lucene Domain Index (Lucene OJVM)

Middle-tier-based Search Engines

Benefits

- Simple (crawler mode)
- Medium complexity (Solr WS)
- Out off the box solution (crawler)
- No application code is necessary to integrate (crawler)
- Medium out off the box solution (Solr WS)

Drawbacks

- Updated are slow (usually monthly, weekly)
- A lot a wasted traffic
- You can not index pages based on database information which requires login (crawler mode)
- Indexing tables requires triggers, batch process or a persistent layer to transfer modifications

Database-embedded Search Engines

Benefits

- Fastest update
- No extra coding its necessary, SQL access
- Ready to use to any language, PHP, Phyton, .Net
- You can index tables
- Changes are automatically notified
- No network traffic
- No network marshalling

Drawbacks

- A little slow down of Java execution compared to a Sun JDK JVM

What is Lucene?

- Open Source Information Retrieval (IR) Library with extensible APIs
- Top level Apache project
- Is the core component of Apache Solr and Nutch projects
- 100% Java
  - Around 800 classes
  - 47.000 lines of code
  - 33.000 lines of test
  - 78.000 lines at ***contrib*** area
- Can search and Index any textual data
- Can scales to millions of pages or records
- Provides fuzzy search, proximity search, range queries, ...
- Wildcards: single and multiple characters, anywhere in the search words

What is Lucene Domain Index?

- An Embebed version of Lucene IR library running inside Oracle OJVM
- 37 new Java Classes and a new PLSQL Object Type
- A new domain index for Oracle Databases using Data Cartridge API (ODCI)
- A new Store implementation for Lucene (OJVMDirectory) which replaces a traditional filesystem by Secure BLOB
- Two new SQL operators lcontains() and lscore()
- An orthogonal/uptodate Lucene solution for any programming language, especially Java, Ruby, Python, PHP and .Net, currently latest production version - 2.3.2.

Benefits

Benefits added to Oracle Applications

- No network round trip for indexing Oracle tables
- A fault tolerant, transactional and scalable storage for Lucene inverted index
- Small Lucene index structure
- Support for IOT
- Support for indexing joined tables using default User Data Store
- Support for indexing virtual columns
- Support for order by, filter by and in-line pagination operations at Index level layer
- Support padding/formatting for Text/Date/Time/Number

But more important than above is

- Easily to adapt for a new functionality

Performance Test Suite

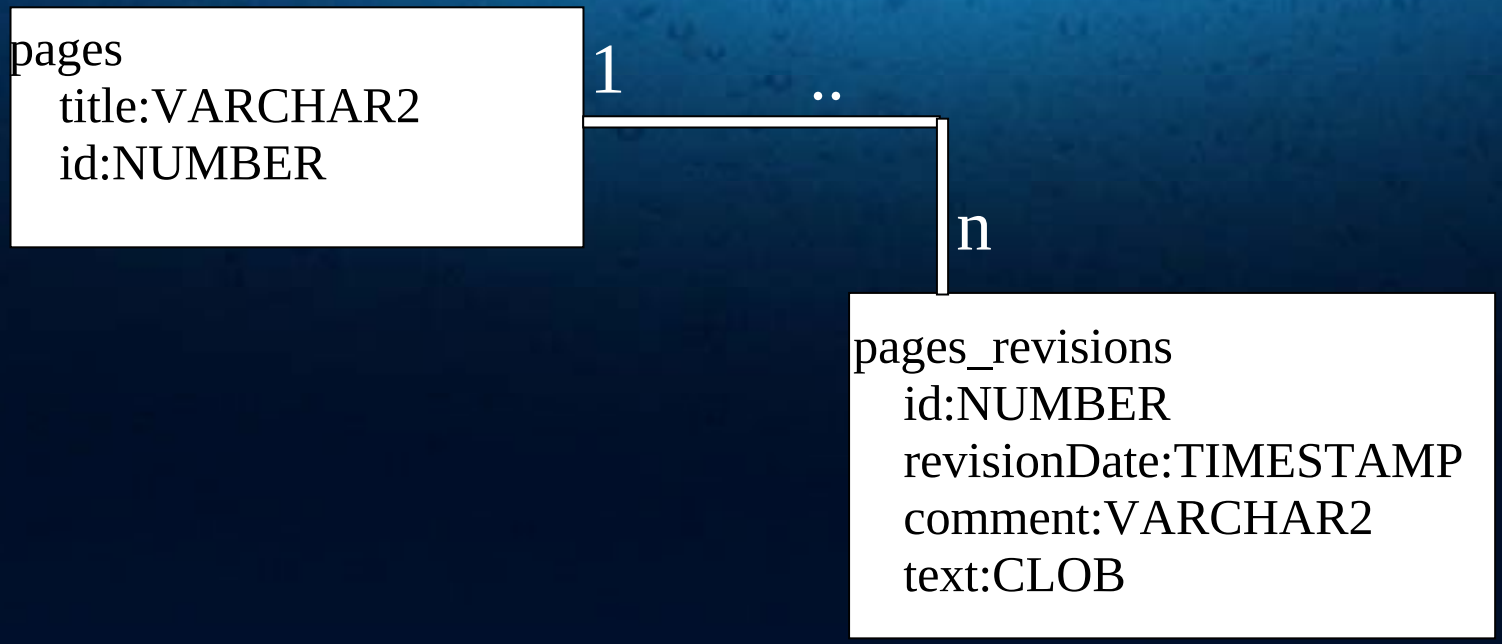
Corpus: XML Spanish WikiPedia dump:

- Total documents: 1.056.163 - 2,67 Gb
- Average size per document: 2.533 bytes

Lucene Index size:

- 10 BLOB/files
- 808Mb total.
- 5 fields (title,revisionDate,comment,text)

Table structure (XMLDB):



Middle-tier-based approach

- Requires transfer all database table data to the middle tier

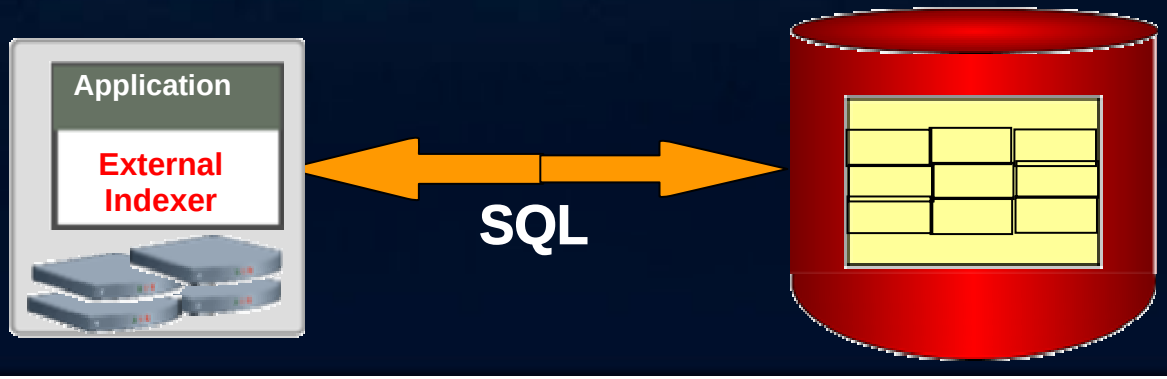
A middle tier application performs this query:

```
SELECT /*+ DYNAMIC_SAMPLING(0) RULE NOCACHE(PAGES) */ PAGES.rowid,
  extractValue(object_value,'/page/title') "title",extractValue(object_value,
  '/page/revision/comment') "comment",extract(object_value,
  '/page/revision/text/text()') "text",extractValue(object_value,
  '/page/revision/timestamp') "revisionDate"
FROM
 ESWIKI.PAGES where PAGES.rowid in
 (select rid from (select rowid rid,rownum rn from ENWIKI.PAGES) where rn>=1 and rn<=300)
```

For ***300*** rows SQLTrace reports:

bytes sent via SQL*Net to client	3.245.358
bytes received via SQL*Net from client	1.785.912
SQL*Net roundtrips to/from client	2.383

Total indexing time for ***33.912*** rows ***824*** seconds



Database-embedded approach

Index Definition, Lucene Domain Index syntax:

```
SQL> ALTER SESSION SET sql_trace=true;
SQL> ALTER SESSION SET EVENTS '10046 trace name context forever, level 8';
SQL> create index pages_lidx_all on pages p (value(p))
indextype is Lucene.LuceneIndex
parameters('PopulateIndex:false;DefaultColumn:text;
SyncMode:Deferred;LogLevel:WARNING;
Analyzer:org.apache.lucene.analysis.SpanishWikipediaAnalyzer;
ExtraCols:extractValue(object_value,"/page/title") "title",
extractValue(object_value,"/page/revision/comment") "comment",
extract(object_value,"/page/revision/text/text()") "text",
extractValue(object_value,"/page/revision/timestamp") "revisionDate";
FormatCols:revisionDate(day);IncludeMasterColumn:false;
LobStorageParameters:PCTVERSION 0 ENABLE STORAGE IN ROW CHUNK 32768
CACHE READS FILESYSTEM_LIKE_LOGGING');
```

Database-embedded approach

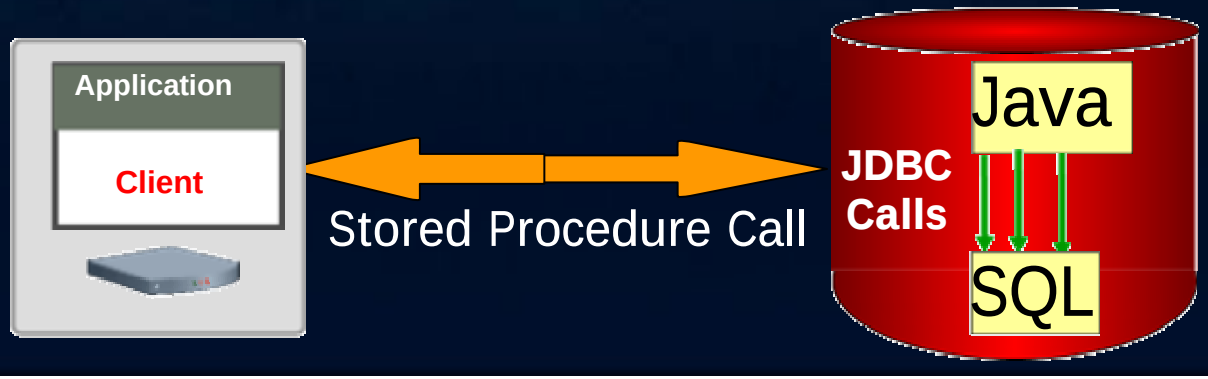
After creating an Index is necessary to submit changes for indexing
This can be done using:

```
DECLARE
  ridlist sys.ODCIRidList;
BEGIN
  select rid BULK COLLECT INTO ridlist
     from (select rowid rid,rownum rn from pages) where rn>=1 and rn<=300;
  LuceneDomainIndex.enqueueChange(USER||'.PAGES_LIDX_ALL',ridlist,'insert');
END;
```

For ***300*** rows SQLTrace reports:

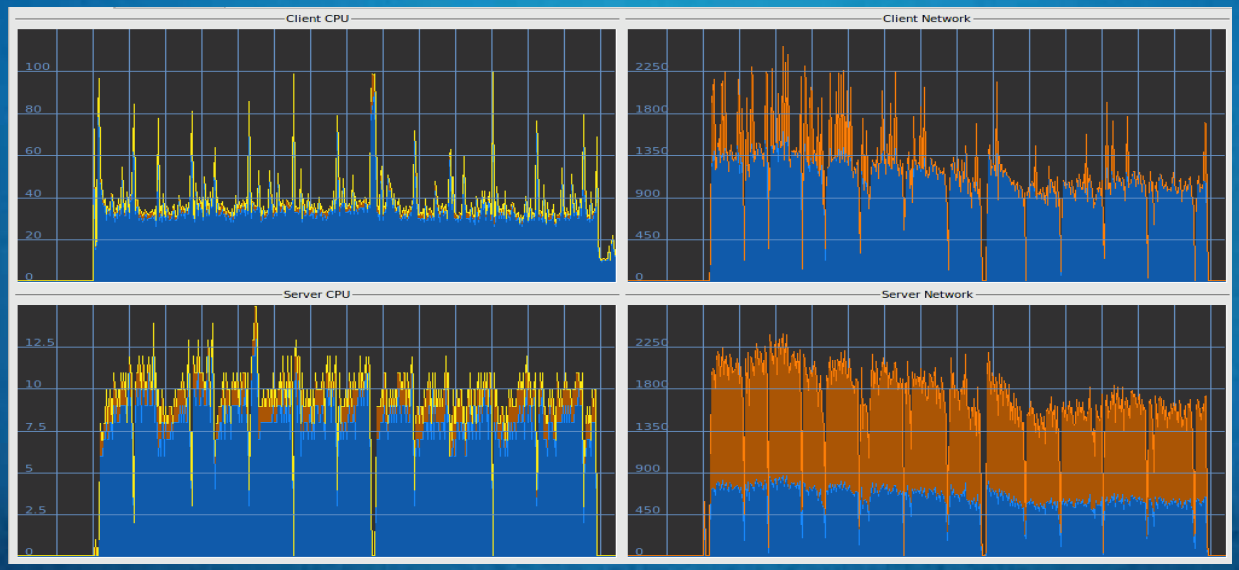
bytes sent via SQL*Net to client	1.301
bytes received via SQL*Net from client	1.354
SQL*Net roundtrips to/from client	4

Total indexing time for ***33.912*** rows ***346*** seconds



CPU / Network usage during indexing

External indexing (824 s.)



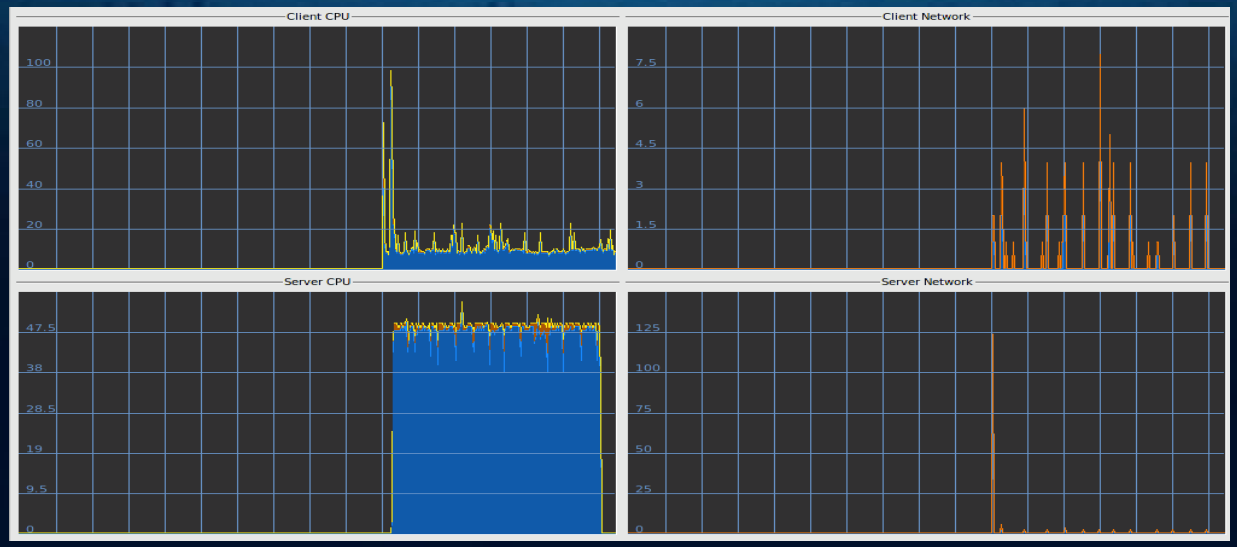
CPU Load
- User
- Sys
- Nice

Client side

Network
- Receiver
- Transmiter

Server side

Integrated indexing (346 s.)



CPU Load
- User
- Sys
- Nice

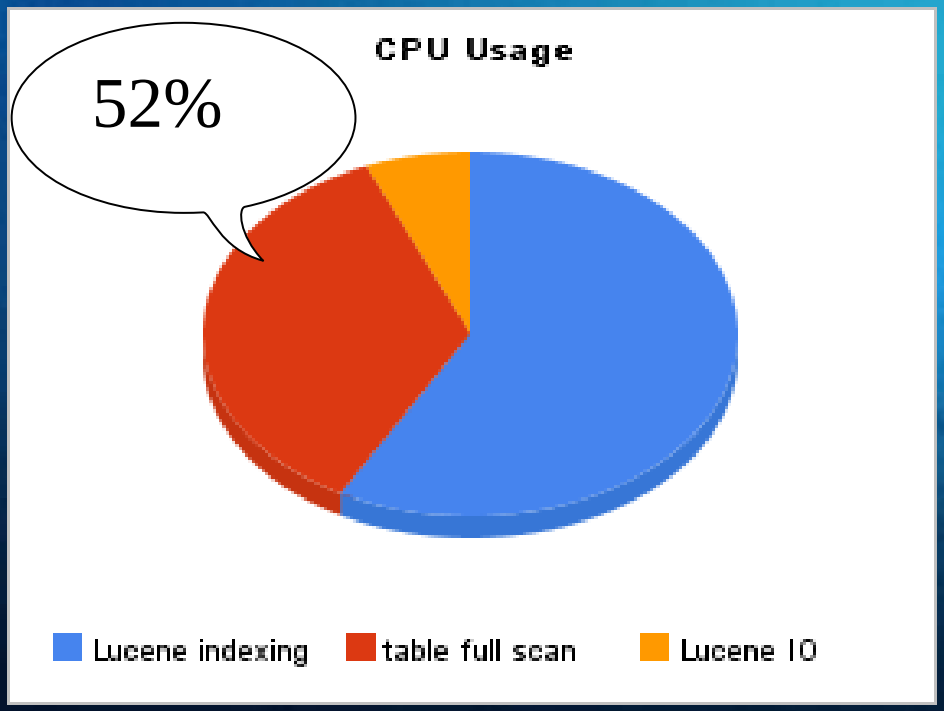
Client side

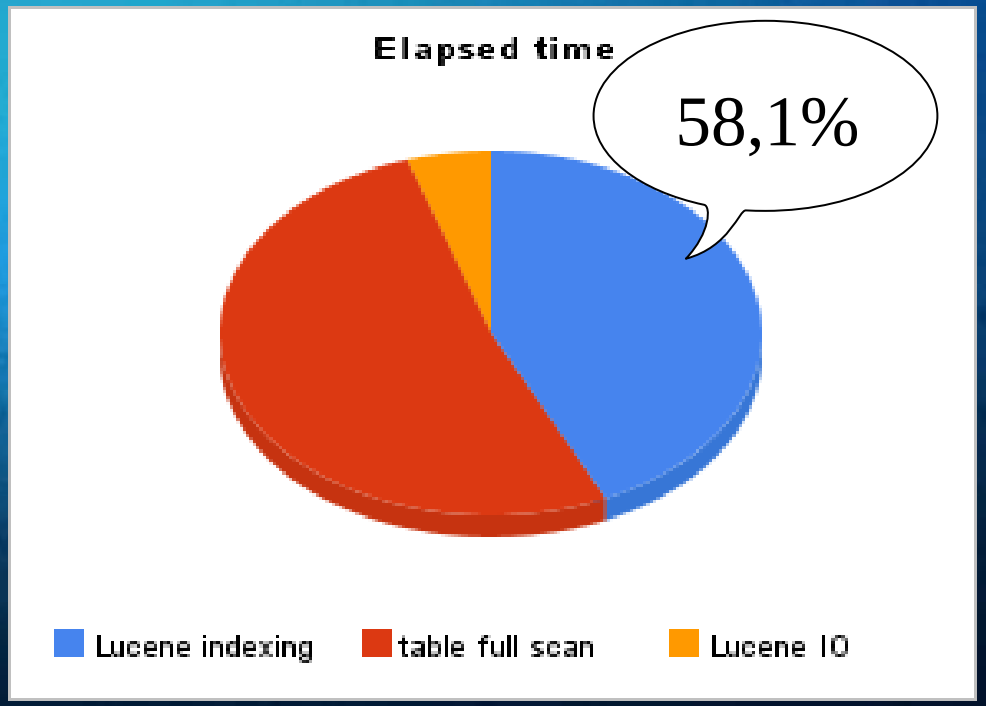
Network
- Receiver
- Transmiter

Server side

CPU/IO for Database-embedded indexing

This information was taken with SQL_TRACE=true indexing 3.000 rows inside OJVM.





Most time is spent in full table scan

In addition, middle-tier indexing of 3.000 rows would require sending 34Mb of data over the network.

Application integration: Fuzzy searches

Old implementation (without Lucene):

```
select p.id,p.first_Name,p.last_Name,p.nationality,p.sex,p.type_Document,
p.number_Document,p.civil_State,p.date_Birth,p.mail,g.organization_id ,
fnmatchperson(p.first_name, p.last_name, 'John Doe') as suma
from person p left join (select * from guest where organization_id = 67) g on g.person_id = p.id
where p.state = 1  and fnmatchperson(p.first_name, p.last_name, 'John Doe') >= 50  order by suma desc
```

Lucene implementation:

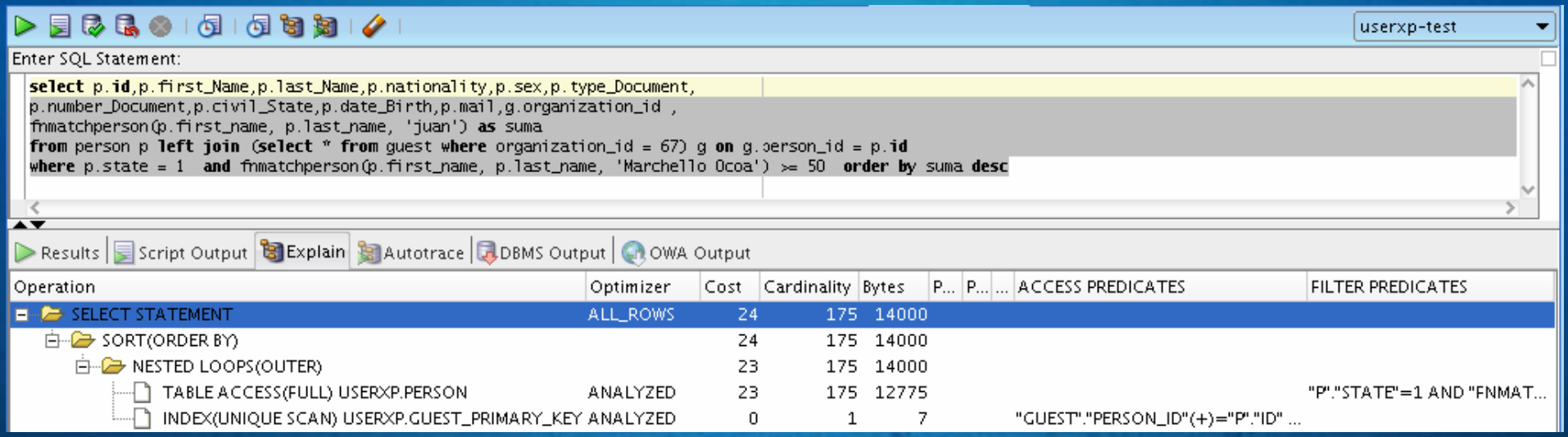
```
select  /*+ DOMAIN_INDEX_SORT */
p.id,p.first_Name,p.last_Name,p.nationality,p.sex,p.type_Document,
p.number_Document,p.civil_State,p.date_Birth,p.mail,g.organization_id ,
lscore(1) as suma from person p left join
(select * from guest where organization_id = 67) g on g.person_id = p.id
where p.state = 1  and lcontains(p.first_name, 'rownum:[1 TO 20] AND John~ Doe~',1) > 0
```

where "***John Doe***" is searched as "***John~ Doe~***" to provide partial match.

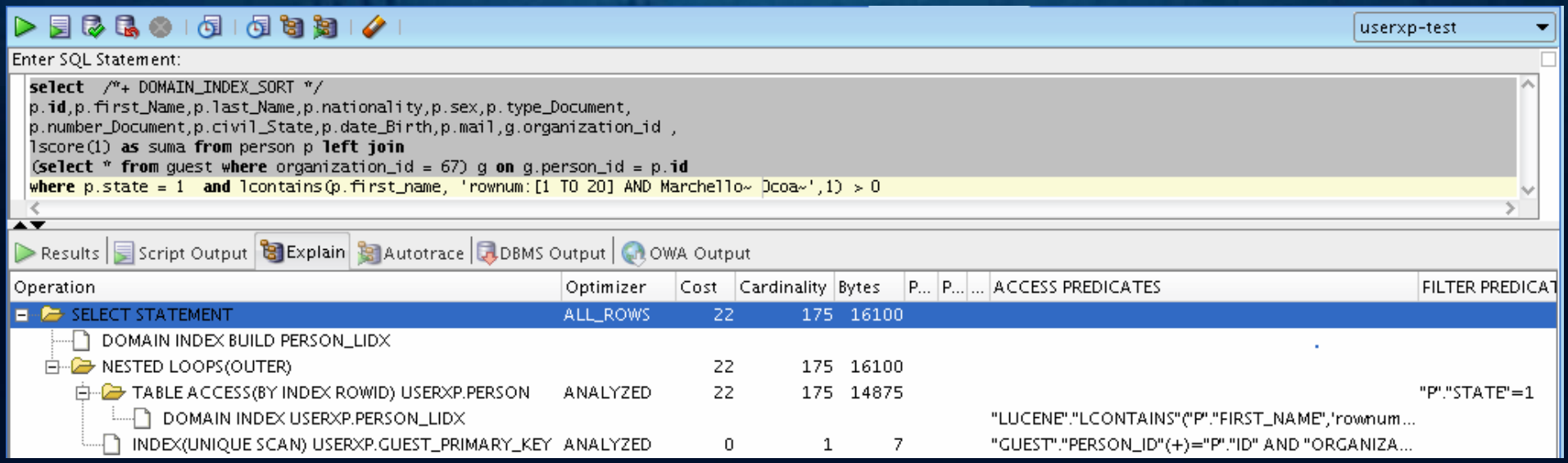
~ Lucene operator uses Levenshtein Distance, or Edit Distance algorithm. http://en.wikipedia.org/wiki/Levenshtein_distance

Execution plan for both queries

fnMatch solution



Lucene solution



Key points

- Only one extra index is required:

```
create index person_lidx on person(first_name)
indextype is lucene.LuceneIndex
    parameters('SyncMode:OnLine;LogLevel:ALL;
AutoTuneMemory:true;IncludeMasterColumn:false;DefaultOperator:OR;
DefaultColumn:name_str;Analyzer:org.apache.lucene.analysis.SimpleAnalyzer;
ExtraCols:first_name||" "||last_name "name_str"');
```

- Simple to adapt, only one class was modified to provide partial match.

```
String split[] = firstLastName.split(" ");
sql3 = "";
for(int i = 0; i < split.length; i++){
sql3 += /*"" +*/ split[i].toLowerCase().trim() + "~ ";
}
sql3 = sql3.substring(0, sql3.length()-1);//le quita la coma
/*sql += ", fnmatchperson(p.first_name, p.last_name, " + sql3 + ") as suma ";*/
/*sql3 = " and fnmatchperson(p.first_name, p.last_name, " + sql3 + ") >= 50 " ;*/
sql3 = "and lcontains(p.first_name, 'rownum:[1 TO 20] AND "+sql3 +"',1) > 0 ";
```

Key points, cont.

- Less network traffic
  - In the above example, around of 20% of the rows are discarded by the filter operation
    - ***"GUEST"."PERSON_ID"(+)="P"."ID" AND ORGANIZATION_ID"(+)=67"***
    - ***"P"."STATE"=1***
  - In Solr implementation a new row on person table imply:
    - N bytes of SQLNet +283 bytes for HTTP Post method
- Faster updates
  - Compared to Solr approach we send 283 bytes less which means faster operations.
  - Compared to middle tier approach, once a new row is added to the table it is ready to be included in the next query in the example shown this is critical constraint
- Minimal application code impact
  - only a new index
  - only a rewrite where condition is needed to replace fnMatch

Future plans

- Add Faceted search, may be using ODCI aggregate functions or Pipeline Tables
- Strong committing to latest Lucene production release, once 2.4 version will be released, we will test inside OJVM
- Add ODCI Extensible Optimizer Interface to better dialogue with the Oracle SQL Engine
- A slave session which collects query from different parallel session to reduce memory foot print and to provides highest hit ratios
- A JMX interface to monitor Lucene Domain Index using Sun's JMX console

Useful links

Lucene Project:

http://lucene.apache.org/java/docs/index.html

Lucene Oracle Integration

http://docs.google.com/Doc?id=ddgw7sjp_54fgj9kg

Forum, Peer to Peer Support

http://sourceforge.net/forum/forum.php?forum_id=187896

Download Binary Distribution (10g/11g)

http://sourceforge.net/project/showfiles.php?group_id=56183&package_id=255524

CVS Access

cvs -d:pserver:anonymous@dbprism.cvs.sourceforge.net:/cvsroot/dbprism login
cvs -z3 -d:pserver:anonymous@dbprism.cvs.sourceforge.net:/cvsroot/dbprism co -P ***ojvm***
http://dbprism.cvs.sourceforge.net/dbprism/ojvm/

Q & A

Thank you