**Resume Search using CTXSYS**

Mark Battle, Devin Horvay, Xihan Qin, Tyler Rust, Ilya Tyagin

University of Delaware

CISC637 - Database Systems

Prof. Bert Gibbons

08 May, 2022

**Introduction**

Companies receive hundreds if not thousands of applications per job opening. When a recruiter or hiring manager receives that many resumes, it’s not feasible for them to carefully read each one, therefore, it would save businesses time, money, and resources to cut down on the manual work a recruiter does in processing resumes.

Since resumes are often stored in relational databases, like Oracle, one option businesses have is to leverage those systems to their full potential. For our proof of concept, we explored the Oracle Text library as a means to easily query resume data based on business needs and return results.

The goal of this paper is to define a number of queries on resume documents using search options within Oracle Text, including CATSEARCH, CONTAINS, and MATCHES, which we demonstrate are viable ways to search and retrieve from unstructured, but common, resume formats like PDFs, Word documents, and even raw text.

### **User Permissions, Schema Creation, and Data Upload**

When it comes to user creation and permissions, this was done for us before beginning the project, but it is worth noting that in order to use Oracle Text, the user must be granted RESOURCE, CONNECT, and CTXAPP roles.

For the schema, the team created four tables.

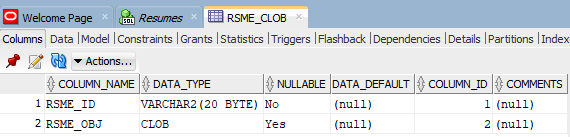
* **RSME** - Table representing a resume that is in BLOB format.
* **RSME\_CLOB** - Table representing a resume that is in CLOB format.
* **CATEGORIES** - Table representing an enumeration of categories, which act as classifiers for raw text.
* **CATEGORIES\_FOR\_SVM\_CLASSIFIER** - Similar to CATEGORIES, but helps improve results for SVM score (further discussion in the MATCHES section)

For data upload, the team worked with Professor Gibbons to upload the resume blobs to the RSME table. In total we used a set of 23 resumes for the blob based searches. We also needed CLOB data for MATCHES search, so we created a custom python script using the cx-Oracle library, which helped us in just a few lines of code to upload 20 resumes in CLOB form.

### **CATSEARCH**

* The operators available for CATSEARCH queries are limited to logical operations. Operators that can be used to define structured criteria are greater than, less than, equality, BETWEEN, and IN.
* CATSEARCH only supports text data types such as VARCHAR2 and CHAR.
* CLOBS and BLOBS are NOT SUPPORTED.
* A CTXCAT index is transactional. When you perform DML (inserts, updates, and deletes) on the base table, Oracle Text automatically synchronizes the index. Unlike a CONTEXT index, no CTX\_DDL.SYNC\_INDEX is necessary. [(Found Here)](https://docs.oracle.com/database/121/CCAPP/GUID-360FFA58-6C94-49FC-B4EC-DA4724CB87C9.htm)

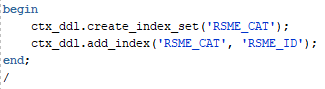
The table I will be working with is RSME\_CLOB.



#### **Step 1:**

**CTX\_DDL.CREATE\_INDEX\_SET(set\_name in varchar2);**

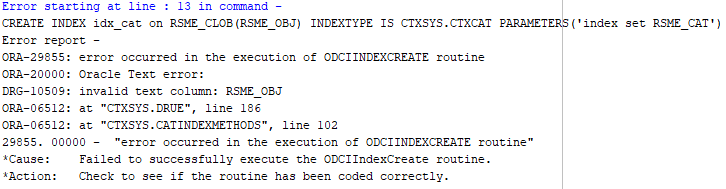
Creates an index set for CTXCAT index types.



To drop an index set, simply use CTX\_DDL.DROP\_INDEX\_SET(‘name’);

**NOTE**: I must use RSME\_ID as the parameter for the index, as it is the only text column available. CLOBs and BLOBs are NOT supported by CATSEARCH.

The following error is produced when trying to index with anything other than a text column.



#### **Step 2:**

**CREATE INDEX ‘index\_name’ on ‘table\_name’(‘text column’) INDEXTYPE IS CTXSYS.CTXCAT PARAMETERS('index set “index\_name” ‘);**

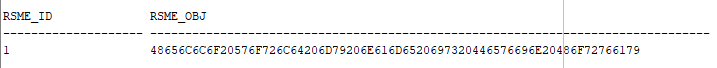
****

#### **Step 3:**

**Query with CATSEARCH**

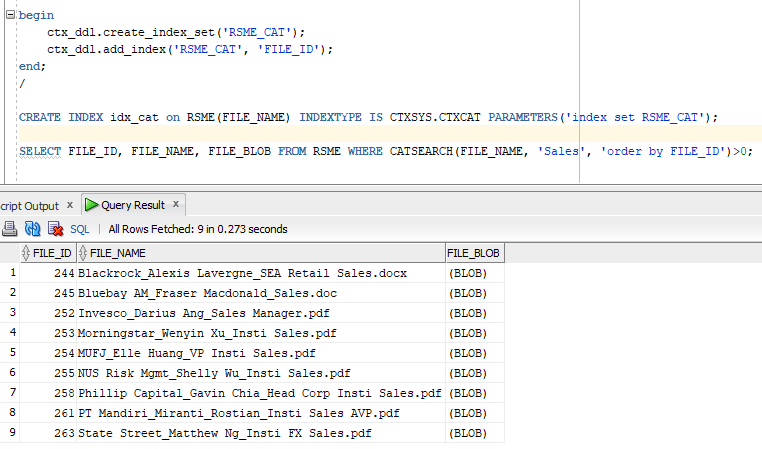
**SELECT \* FROM ‘table\_name’ WHERE CATSEARCH(‘index parameter’ , ‘query\_string’, ‘order by “column\_name” ‘) ‘operator’ ‘value’;**

****

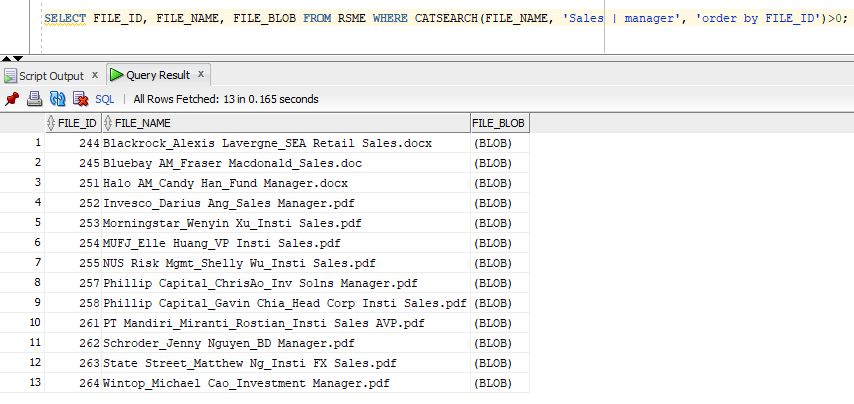


#### **End of Steps.**

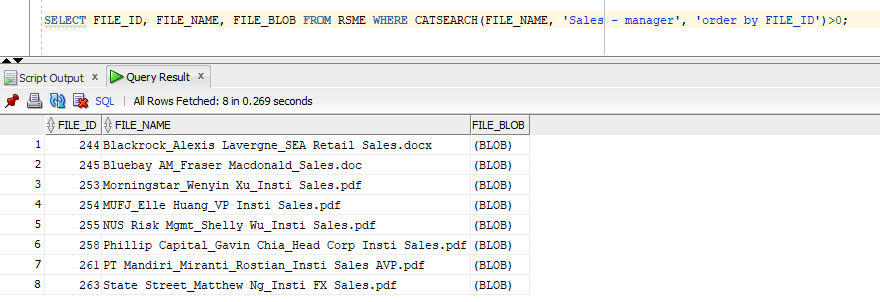
Now that the fundamental implementation of CATSEARCH is established, it can be demonstrated on our resume dataset.



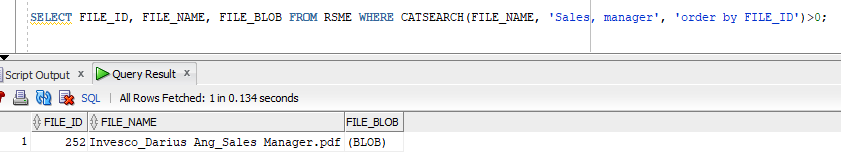
#### **CATSEARCH with OR**



#### **CATSEARCH with NOT**

****

#### **CATSEARCH with AND**

****

### **CONTAINS and SCORE**

Before searching with CONTAINS and SCORE, CONTEXT index needs to be created. CONTEXT index type is good for indexing large, coherent documents in the formats as Microsoft Word, PDF, HTML, or plain text. CONTEXT index is not transactional. If a record is deleted, CONTEXT index will change immediately. The search results will no longer include the deleted record. If a record is inserted or updated, an index synchronization is needed to make the new record be seen for searching.

CONTAINS is used in the SELECT statement to specify the query in the WHERE clause. The CONTAINS operator must always be followed by the >0 syntax, which specifies the score value returned by CONTAINS operator. When the CONTAINS score >0, result rows will be returned.

When the SCORE operator is called in the SELECT statement. The CONTAINS operator must reference the score label in the third parameter.

There are different logical operators that can be used to limit the search criteria including AND, OR, NOT, ACCUM, and EQUIV.

Sample code, query and results are provided below.

#### **Create CONTEXT index**

After the table RSME is created and resumes are uploaded. create CONTEXT index.

****

#### **Sync CONTEXT index**

Professor Gibbons helped to remove the first batch, created resume table and uploaded new resumes.

For this case, the index does not automatically sync itself. We need to sync index.

And ctx\_ddl needs to be granted before use.

“GRANT EXECUTE ON CTXSYS.CTX\_DDL TO UD\_CISC637\_GROUP4;”

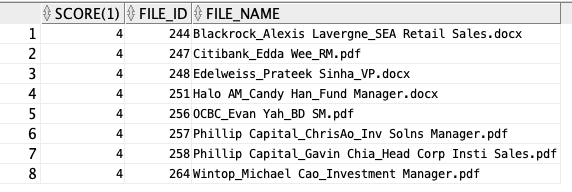
****

#### **Search from blob**

Below are the example queries and results for both CONTAINS and SCORE

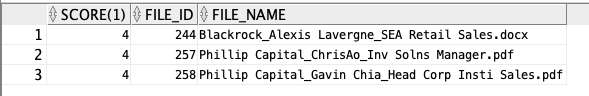
##### 3.1 **Search one term with SCORE operator**

****

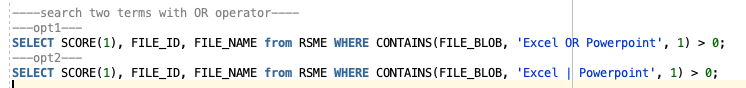
****

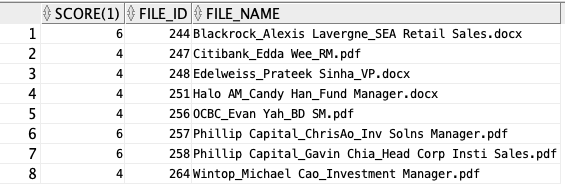
##### 3.2 **Search two terms with and operator and SCORE operator**

****

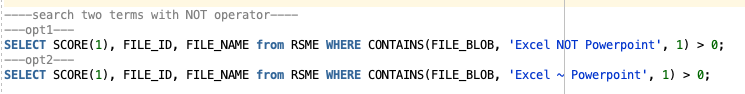
****

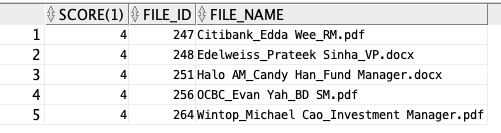
##### 3.3 **Search two terms with or operator and SCORE operator**

****

****

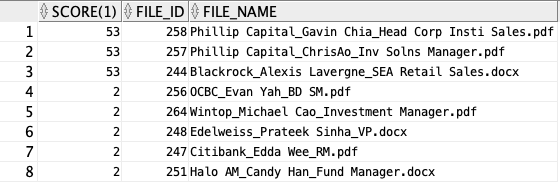
##### 3.4 **Search two terms with not operator and SCORE operator**

****

****

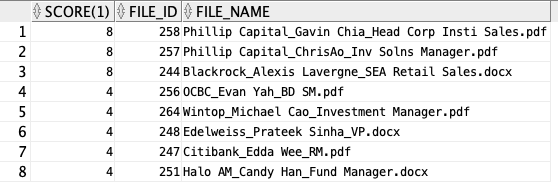
##### 3.5 **Search two terms with ACCUM operator and SCORE operator**

****

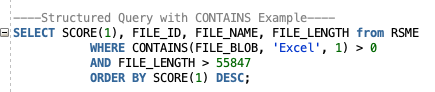
****

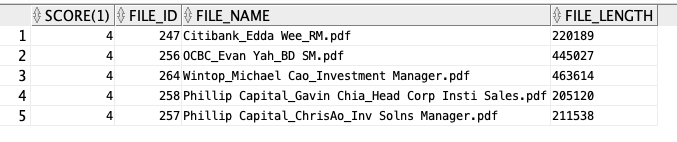
##### 3.6 **Search two terms with EQUIV operator and SCORE operator**

****

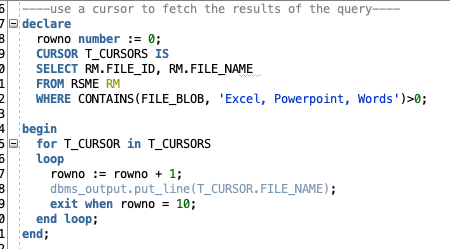
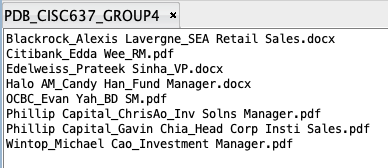
****

##### 3.7 **Structured Query with CONTAINS and SCORE Example**

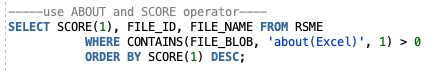
****

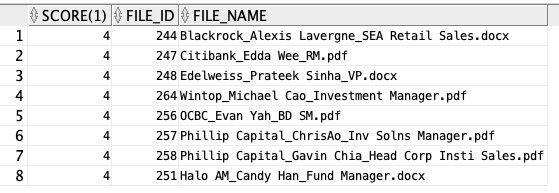
****

##### 3.8 **Use a cursor to fetch the results of the query**

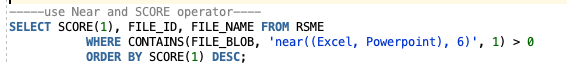
****

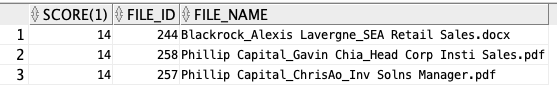
##### 3.9 **Use ABOUT and SCORE operator**

****

****

##### 3.10 **NEAR operator and SCORE operator**

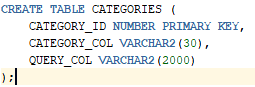
****

****

### **MATCHES and MATCH\_SCORE**

#### **Create Table to Hold Resume CLOBS** For MATCHES, you cannot use BLOBS. Therefore, you must have a table (separate from the table with Resume BLOBS) that holds VARCHAR2 or CLOBs. These can be notes or documents on candidates or their resume. The document can be plain text, HTML, or XML. Here are some filled in rows in the table (there is more text, but just showing what the CLOBS look like):

#### **Create Categories Table And Insert Columns**

In order to use MATCHES, you must have a table of queries that you can use to categorize each CLOB document. You need a table that stores categories based on keywords. This makes sense if you have common categories you would like to label your applicants under. One example is a big company that gets resumes and wants to categorize them under different jobs or skills. You could look for keywords about oracle or software engineering and categorize them under the full stack developer category whereas one who talks about logistics would be under a business category.  
This table is where you use logical operators to be able to categorize documents based on queries.   


##### **2.1 Using NEAR Operator**

****

****

##### **2.2 Using OR Operator**

****

##### **2.3 Using NOT Operator**

****

##### **2.4 Using ABOUT Operator**

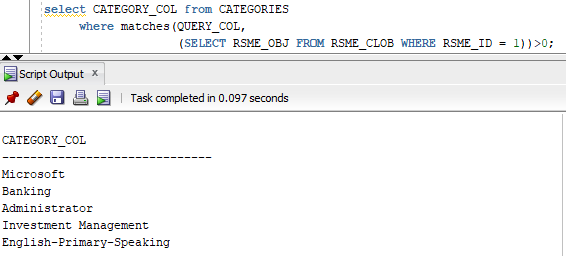
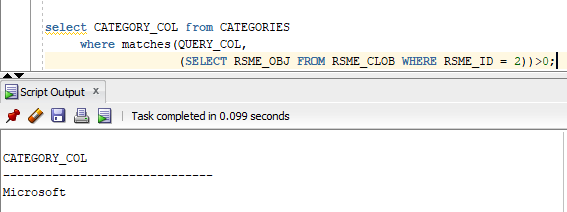
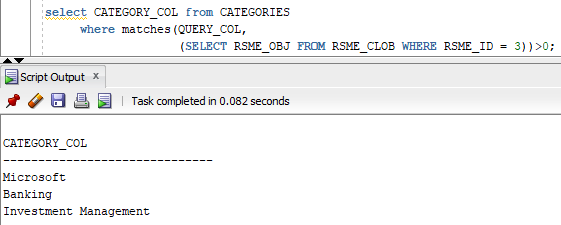
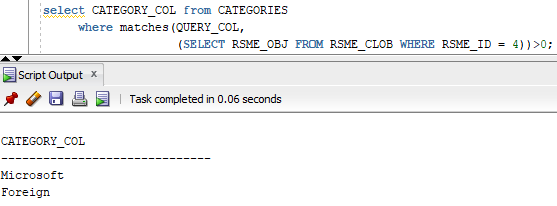
****

****

#### **Create CTXRULE Index on Categories Table**

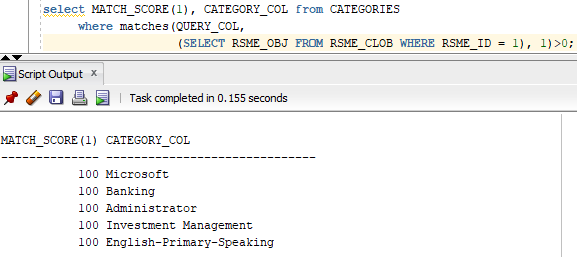
The CTXRULE index is essentially an index on the set of queries that define your classifications.



1. **Using MATCHES**The best way to use the MATCHES operator is on incoming documents. The best idea is to create a trigger as documents are placed into the table to categorize them. However, to demonstrate how MATCHES works, I will just select one CLOB at a time from our table to categorize based on our CATEGORIES table.  
   Below are several examples of running the MATCHES command on different CLOBS in the RSME\_CLOB table:  
     
     
     
   
2. **Using MATCH\_SCORE With MATCHES**The MATCH\_SCORE operator is used in a statement to return scores produced by a MATCHES query.

##### **5.1 Using MATCH\_SCORE Without SVM\_CLASSIFIER**

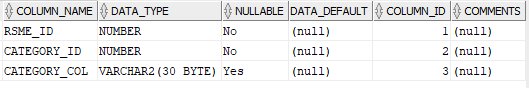
The output is as follows using MATCH\_SCORE:

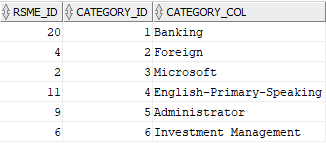
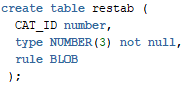
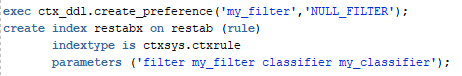
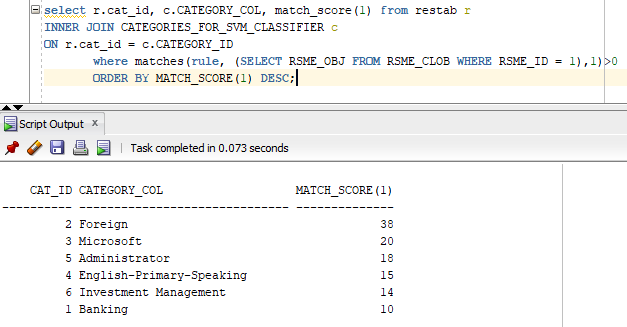
  
Essentially, this is going to return a 100 if the matches operator returns a match to any of the categories. This does the same thing as the MATCHES operator.

##### **5.2 Using MATCH\_SCORE With the SVM\_CLASSIFIER**

The way to improve MATCH\_SCORE is to use the SVM\_CLASSIFIER as a preference. For this to work, you have to train the classifier by having documents with predefined categories. The SVM\_CLASSIFIER can then read those and begin to label new documents as they come in. This gives a true output of how closely related each new document is to a category based off of predefined document-category pairs.

The first step is to create a new table with a foreign key pointing to the RSME\_CLOB table. I called this new table CATEGORIES\_FOR\_SVM\_CLASSIFIER.



The next step is to add document-category pairs to the table. I tried to keep the categories the same as before adding documents that only really matched one category to that category. I came out with the following:  
  
Next I had to create a context index on the RSME\_CLOB table:  
  
Followed by setting the preferences to be the SVM\_CLASSIFIER (instead of using the rule classifier as I had been using in the above MATCHES queries):  
  
Then, in order to train the classifier, I had to create the exact table shown below:  
  
That is the results table. I then ran the following command to train the classifier:  
  
Finally, I can recreate the CTXRULE index with the parameter being the classifier I just trained:  
  
Then, I can run the MATCH\_SCORE with the MATCHES query in order to see how close a match each document is to each category:  
  
This is what the MATCH\_SCORE should really do; rate how closely each document is matching a category.

### **INDEX SYNCHRONIZATION BEHAVIOR**

Oracle Text domain indices come in two forms: transactional and non-transactional. In this context, non-transactional means that new information is not visible to text searches until an index synchronization has occurred. For non-transactional indices, the user must explicitly run the synchronization query CTX\_DDL.SYNC\_INDEX before their searches contain newly inserted or updated rows; however, if a row is deleted the index is changed immediately and that row becomes unavailable to search for both your session and other users. Transactional indices do not have this limitation - whenever a row is inserted, updated, or deleted, the index is reindexed. The CONTAINS index is non-transactional while the CTXCAT is transactional.

When documents in the indexed table are inserted, updated, or deleted, their row ids are held in a DML queue until you synchronize the index, which can be queried from CTX\_USER\_PENDING. Once the CTX\_DDL.SYNC\_INDEX query is run, this queue is processed and all rows are then available for searching.