INTRODUCTION
SAS® is one of the most versatile software packages available on the market today; with it you can analyse everything from genetics to market research, financial to quality and risk data all using variations of SAS. However, with that versatility comes a price - sometimes there is something you absolutely have to do and cannot seem to find an easy way of doing it. When that happens, it is as if the earth stops; "What do you mean I can't do this in SAS?" you ask yourself. You spend sleepless nights online, looking at every paper Lex Jansen has, convinced someone has done what you're trying to accomplish¹. In some cases you may find that obscure paper with the lines of code you have been so desperately searching for. Sometimes, you don't - then comes a hard decision; do you figure out a way of accomplishing your seemingly impossible task in SAS® or do you (gasp) go elsewhere? This was the decision I was forced to make, and I went with SAS® (obviously, because I am writing about it!).

The dilemma - how to make a data dictionary for an Oracle database, where PROC Contents will not work (I am using SAS/ACCESS, and SAS® doesn't recognize the virtual library created by the LIBNAME statement to pull a contents report).

The solution - use Oracle's system tables and pull the data into SAS.

BACKGROUND
I am the database administrator for one of the largest pediatric hospitals in North America; the program I work for is the largest of its kind in all of Canada. Our database, deployed in 2002, has not had any major upgrades despite medical and process improvements and changes over the decade. As a result, we have had an increasing number of fields that the users no longer fill in, which has obvious clinical and research implications. Being in my role for about a year and a half when I undertook this project, and coming from a previous job that was very fastidious about documentation about our group's database, I thought this was something that would take me a week. Try 6 months of emails, reading, coding/recoding/deleting/starting all over again, and way too much coffee. In hindsight, this code is not as complex as I thought at the time; however, I had been using SAS® for about a year, and most of that was PROC SQL. I had no idea about ODS, FORMAT, PROC TEMPLATE, or any of the other SAS® skills I would acquire along the way. But, I felt it was important to present this because chances are good there is someone out there that is in the same boat I was.

CODE
The first section of the code develops the template that I wanted to use, allowing for the titles to be specific fonts / font sizes and formatting.

```
proc template;
define style MyLib;
parent = styles.analysis;

class systemtitle /* Database Data Dictionary */
font_face="arial"
font_size=5
font_weight= bold
foreground=darkblue
background=white;
class title /
font_face="arial"
font_size=3.5
foreground=darkblue;
class header /* Table headers */
font_face="Arial"
font_size= 7pt
foreground=chocolate
just=center
vjust=middle;
class data /* Rows in the tables */
```

¹ This may also be my own personal experience…
As you can see I'm specifying different formats for the Document header, table header, and rows in the table. I admit, some of this was just for my own learning, but after I saw how it looked I decided to stick with it.

The next section goes through some rather specific database administrative / performance enhancement data, including tables that do not have indexes, tables that have not been analysed in more than xxxx number of months, etc. These were all contained in PROC SQL and I've removed some lines of code that contained confidential information. You'll notice that I'm using both "AS" and "LABEL". This is because when I used just "AS", pulling in from our Oracle database the columns did not get the label set up; so I'd have NAME=Avg_Space but LABEL=VAR1. When I realized what was happening, I was really annoyed and it took me quite a while to fix.

CREATE TABLE datadictionary as
select
t.TABLE_NAME as Table label='Table',
t.LAST_ANALYZED as Analyzed label='Analyzed',
NUM_ROWS as Rows label='Rows',
BLOCKS as Blocks label='Blocks',
EMPTY_BLOCKS as EmptyBlocks label='EmptyBlocks',
AVG_SPACE as Avg_Space label='Avg_Space',
AVG_ROW_LEN as Avg_Row_Len label='Avg_Row',
t.SAMPLE_SIZE as Sample label='Sample',
ROW_MOVEMENT as RowWvmt label='RowWvmt',
t.GLOBAL_STATS as Global_Stats label='Global_Stats',
COLUMN_NAME as Column label='Column',
DATA_TYPE as Data_Type label='Data_Type',
AVG_COL_LEN as Avg_Col_Len label='Avg_Col',
CHAR_COL_DECL_LENGTH as Decl_Len label='Decl_Len',
CHAR_LENGTH as Char_Len label='Char_Len',
DATA_DEFAULT as Data_Dflt label='Data_Dflt',
DATA_LENGTH as Data_Len label='Data_Len',
DATA_PRECISION as Data_Prec label='Data_Prec',
DATA_SCALE as Data_Scale label='Data_Scale',
DEFAULT_LENGTH as Dflt_Len label='Dflt_Len',
DENSITY as Density label='Density',
HISTOGRAM as Histogram label='Histogram',
INITIAL_EXTENT as Init_Extent label='Init_Extent',
NULLABLE as NullYN label='NullYN',
NUM_BUCKETS as Num_Buckets label='Num_Buckets',
NUM_DISTINCT as Num_Distinct label='Num_Distinct',
NUM_NULLS as Num_Nulls label='Num_Nulls'
from MyLib.all_tables t, MyLib.all_tab_columns c
where t.table_name=c.table_name
order by t.table_name;

I should also note that in the WHERE clause, you can set very specific parameters; in my case, I have TABLESPACE_NAME and OWNER set to certain values as those tables / columns are the only ones I want (removed due to privacy concerns).

However, being somewhat of a Database Geek, I also wanted to look at possible areas of performance issues, missing indexes, etc. Reading through the voluminous Oracle documentation, I found that there are queries you can run that will provide you with more detail that you'll know what to do with. I don't know precisely all what these queries do, but I understand the important part – they are telling me there are problems. I sent the list of results over to the Database Administration team in my organization and they have been working to correct the major issues for about a month. These queries were taken from http://vsbabu.org/oracle/sect16.html and then modified according to my requirements. The descriptions below are my interpretations of what the author of the website provided, so any.
errors are mine. As with all other code in this paper, I have removed certain sections that contain confidential or other sensitive information.

Query 1: Questionable Indexes

CREATE TABLE QuestionNDX as
select column_name, table_name
from MyLib.all_ind_columns
where column_position=1
group by table_name, column_name
having count(*)>1;

This query highlights all tables that have more than one index with the same leading column, which can result in queries using an incorrect index. Put another way, Oracle will use the index that was created most recently if two (or more) indexes are of equal ranking. This could potentially result in different indexes being used from one environment to the next (for example, from Development to Test to Production). The author of the website warns that this does not automatically indicate that there is an issue, but just something that you need to be aware of and that you may end up needing to justify the existence of the indexes.

Query 2: Tables with Multiple indexes

CREATE TABLE MultiNDX1 as
select OWNER,
    TABLE_NAME,
    COUNT(*) AS index_count
from MyLib.all_indexes
group  by OWNER, TABLE_NAME
having COUNT(*) > 5
order  by OWNER, TABLE_NAME;

CREATE TABLE work.MultiNDX as
select ai.index_name, ai.table_name, ai.uniqueness,
ai.distinct_keys, ai.num_rows, ai.sample_size,
ai.last_analyzed, ai.generated
from work.multindx1 as m, MyLib.all_indexes as ai
where m.table_name = ai.table_name
order by ai.table_name, ai.index_name;

The original query does this in one, where I’ve broken it down to two; I did this because I wanted to see columns from the ALL_INDEXES table that had I included in the first query would have caused duplicate rows. You can also change the HAVING parameter in the first query to be any value you want; I shudder to think that there would be more than 5 indexes on a table, but I suppose it’s possible. Play around with it in your database and see what you come up with.

Query 3: No Indexes

CREATE TABLE work.NoNDX1 as
select t.TABLE_NAME
from  MYLIB.ALL_tables t left join MyLib.all_indexes i on t.table_name =
i.table_name
group t.table_name
having count(i.table_name)=0
order  by t.TABLE_NAME;

CREATE TABLE work.nondx as
select n.table_name, c.column_name
from work.nondx1 n, MyLib.all_tab_columns c
where n.table_name = c.table_name;

In the first query, you could have a WHERE statement specifying only certain Databases / Owners that you’re interested in. As I have it set up right now, this will pull all databases – potentially a very large report.

Query 4: No Primary Key

CREATE TABLE work.noPK as
Select TABLE_NAME
from MYLIB.all_tables dt
where not exists (  
    select 'TRUE'  
    from MYLIB.ALL_constraints dc  
    where dc.TABLE_NAME = dt.TABLE_NAME  
    and dc.CONSTRAINT_TYPE='P')  
order by OWNER, TABLE_NAME;

As any good Database Administrator will tell you, Primary keys (or the Unique Identifier for a table) are critical to a database performing well. They are used to ensure data is not duplicated, and provide some stability to the table structure. The Constraint_Type = 'P' is the flag for Primary key; the Oracle documentation has other Constraint Types you can specify.

Query 5: Foreign Keys without an Index

CREATE TABLE work.FKnoNDX as  
select ACC.TABLE_NAME,  
    acc.CONSTRAINT_NAME,  
    acc.COLUMN_NAME,  
    acc.POSITION  
from   MyLib.all_cons_columns acc,  
MyLib.ALL_constraints ac  
where ac.CONSTRAINT_NAME = acc.CONSTRAINT_NAME  
and   ac.CONSTRAINT_TYPE = 'R'  
and   acc.OWNER= 'PAMOT'  
and not exists (  
    select 'TRUE'  
    from MyLib.ALL_IND_columns b  
    where b.TABLE_OWNER = acc.OWNER  
    and b.TABLE_NAME = acc.TABLE_NAME  
    and b.COLUMN_NAME = acc.COLUMN_NAME  
    and b.COLUMN_POSITION = acc.POSITION)  
order by acc.OWNER, acc.CONSTRAINT_NAME, acc.COLUMN_NAME, acc.POSITION;

Foreign Keys are a group of columns that combined make up a Primary key; this would ensure, for example, that only one Clinic Date is entered for each patient (Date+Patient ID). However, having a foreign key without an index on the child table could prove to be problematic. It is highly recommended that an index be created if the foreign key will be used in joining, or is going to be frequently used in WHERE clauses. A table-level lock will be placed on the parent table, causing freezing and potentially corrupting the database.

Query 6: Columns with the Same Names but Different Lengths

CREATE TABLE DiffLens1 as  
SELECT A.COLUMN_NAME  
FROM MyLib.ALL_TAB_COLUMNS A, MyLib.ALL_TAB_COLUMNS B  
WHERE A.COLUMN_NAME=B.COLUMN_NAME  
AND A.TABLE_NAME = B.TABLE_NAME  
GROUP BY A.COLUMN_NAME  
having (min(A.DATA_LENGTH)- max(B.DATA_LENGTH))<>0  
ORDER BY A.COLUMN_NAME;

CREATE TABLE DiffLens2 as  
select table_name, dl.column_name, a.data_type, a.data_length  
from MyLib.all_tab_columns a, work.difflens1 dl  
where a.column_name = dl.column_name  
order by dl.column_name;

CREATE TABLE DiffLensMaxMin as  
select dl.column_name, max(a.data_length) as Max1, min(a.data_length) as Min1  
from MyLib.all_tab_columns a, work.difflens1 dl  
where a.column_name = dl.column_name  
GROUP BY dl.column_name  
ORDER BY dl.column_name;

CREATE TABLE work.DiffLensFinal as  
select d.*,  
    FOUNDATIONS AND FUNDAMENTALS  
NESUG 2012
Just to quickly walk through these queries, the first is to get the column names where I’ve joined the
ALL_TAB_COLUMNS table onto itself. I have then joined the table onto itself by Table and Column Name, and in my
HAVING clause, said include only those where the difference is not equal to 0.
In the second section, I want to get the column names from the first table and then get specific variables (Data Type,
Data Length, Table Name) that I want to see.
The third section sets up the range for the Data Length per column, and the final one puts it all together into a single
output.

I have now created all the tables I require for my rather extensive data dictionary; depending on the Oracle database
you are working in, you may get 0 rows back so it'll be a short document in the end; or, you can have a situation
similar to mine where to print the document would effectively require a small forest.

The next step is to compile the various tables into the final report; this is done using PROC REPORT, which I won’t
go through in any sort of detail other than to highlight a couple of cool features I learnt along the way.

The first was COMPUTE COLUMN, which I used like this:
compute column;
  if column in (‘Column_1’,’Column_2’) then do;
    call define(_col_,’style’,’style=[background=plum]’);
  end;
endcomp;

For this particular output, any columns I specify (in my case, I wanted my Patient Medical Record number) highlighted
in a plum colour so it stands out from the rest of the document.

Within the DEFINE statement, I learnt that there are a number of items you can specify. I won’t go into all of them,
but here are some of the more important ones I wanted to highlight:
DEFINE column_name / noprint;
The column will not print out in the display. In my report, I did not want MIN1 and MAX1 for one of
the columns to show up but I needed them for colour-coding logic; the NOPRINT did the trick.
DEFINE variable / order;
Sorts the table by whatever variable you’ve specified.
DEFINE table / across;
The unique values of a variable each get their own column; in other words, it’s a simple way to put
everything into a cross tab.

Finally, the aim was to have an Excel spreadsheet that would allow for filtering and sorting by columns (for example,
ones that have NULLs more than 75% of the table) but we also wanted something that would be easily printed and
used by the researchers in our program to determine if a variable was not only in the database, but also how often it
was being populated. I should note here that after the generation of this Word document, I spent another week or so
reviewing the data and reworking some of the columns; I merged some cells, added a “Recommended data source”
column for some of the tables that had a higher incidence of missing data, and added in the field type column and the
options available if the variable was a drop box. Because of the way our database was set up, the options for the
drop boxes are hardcoded into the form; as a result, there is no way I could extract this data using SAS® or any other
software package (that I’m aware of).

Here’s a screen shot of some of the finalized reports; excuse the blurred portions, they are necessary for proprietary
reasons.

The Excel File, with just the data dictionary:
Figure 1: A portion of the Data Dictionary in Excel

Below is a screen shot of the first table that is in the above screen shot, but taken from the Word Document. Note the different level of detail, and the plum-coloured cell third from the bottom that indicates the variable(s) I specified in the COMPUTE above.

Figure 2: The Formatted Data Dictionary in Word

Here are some screen shots of the various Database Performance queries that were discussed above.

Figure 3a and b: Portion of the Performance Tuning Queries
CONCLUSIONS
Having a Data Dictionary that lists all Tables, Variables, and metadata is a key tool for Researchers, Analysts, or Users. However, if the database is in Oracle, creating one is not the easiest task; one is required to have extensive knowledge of the System Tables, and how to extract usable data from them. This paper is an attempt to show how one is created with minimal amounts of SAS® Code (less than 300 lines) but is a powerful document that is now being used program-wide.

ACKNOWLEDGEMENTS
I’d like to thank the section chairs for allowing me to present, to the SAS-L and SAS® Canada Communities for their support in getting through this code, and to all the people I’ve had the opportunity and pleasure to chat with about this project.

Although the complete code cannot be provided, please contact me below if you are having trouble with your Oracle Data Dictionary.

TRADEMARK CITATION
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION
Christopher Battiston,
Solid Organ Transplant Program,
SickKids
555 University Avenue,
Toronto, ONT
M5G 1X8

Email: darth.pathos@gmail.com

REFERENCES

