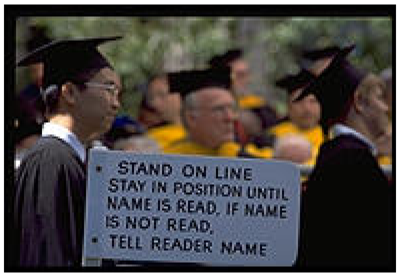
Escaping to the procedural world

5 min. read · View original



Declarative languages can be very powerful and reliable, but sometimes it is easier to think about things procedurally. One way to do this is by using a procedural language in the database client. For example, with AOLserver we generally program in Tcl, a procedural language, and read the results of SQL queries. For example, in the /news module of the ArsDigita Community System, we want to

- query for the current news
- loop through the rows that come back and display one line for each row (with a link to a page that will show the full story)
- for the first three rows, see if the news story is very short. If so, just display it on this page

The words above that should give a SQL programmer pause are in the last bullet item: *if* and *for the first three rows*. There are no clean ways in standard SQL to say "do this just for the first N rows" or "do something special for a particular row if its data match a certain pattern".

Here's the AOLserver Tcl program. Note that depending on the contents of an item in the news table, the Tcl program may execute an SQL query (to figure out if there are user comments on a short news item).

```
set selection [ns_db select $db "select *
from news
where sysdate between release_date and
expiration_date
and approved_p = 't'
order by release_date desc, creation_date desc"]

while { [ns_db getrow $db $selection] } {
    set_variables_after_query
    # we use the luxury of Tcl to format the date
nicely
    ns_write "<li>[util_AnsiDatetoPrettyDate
$release_date]: "
    if { $counter <= 3 && [string length $body] <
300 } {
        # it is one of the top three items and it
is rather short
        # so, let's consider displaying it right
here
        # first, let's go back to Oracle to find
out if there are any
        # comments on this item
        set n_comments [database_to_tcl_string
$db_sub "select count(*) from general_comments
where on_what_id = $news_id and on_which_table =
'news'"]
        if { $n_comments > 0 } {
            # there are some comments; just show
the title
            ns_write "<a href=\"item.tcl?
news_id=$news_id\">$title</a>\n"
        } else {
            # let's show the whole news item
            ns_write
"$title\n<blockquote>\n[util_maybe_convert_to_html
$body $html_p]\n"
            if [ad_parameter SolicitCommentsP
news 1] {
                ns_write "<br><br>\n<A
HREF=\"comment-add.tcl?
news_id=$news_id\">comment</a>\n"
            }
            ns_write "</blockquote>\n"
        }
    } else {
        ns_write "<a href=\"item.tcl?
news_id=$news_id\">$title</a>\n"
    }
}
```

Suppose that you have a million rows in your news table, you want five, but you can only figure out which five with a bit of procedural logic. Does it really make sense to drag those million rows of data all the way across the network from the database server to your client application and then throw out 999,995 rows?

Or suppose that you're querying a million-row table and want the results back in a strange order. Does it make sense to build a million-row data structure in your client application, sort them in the client program, then return the sorted rows to the user?

Visit http://www.scorecard.org/chemical-profiles/ and search for "benzene". Note that there are 328 chemicals whose names contain the string "benzene":

```
select count(*)
from chemical
```

```
where upper(edf_chem_name) like
upper('%benzene%');

  COUNT(*)
----------
       328
```

The way we want to display them is

- exact matches on top
- line break
- chemicals that start with the query string
- line break
- chemicals that contain the query string

Within each category of chemicals, we want to sort alphabetically. However, if there are numbers or special characters in front of a chemical name, we want to ignore those for the purposes of sorting.

Can you do all of that with one query? And have them come back from the database in the desired order?

You could if you could make a procedure that would run inside the database. For each row, the procedure would compute a score reflecting goodness of match. To get the order correct, you need only ORDER BY this score. To get the line breaks right, you need only have your application program watch for changes in score. For the fine tuning of sorting equally scored matches alphabetically, just write another procedure that will return a chemical name stripped of leading special characters, then sort by the result. Here's how it looks:

```
select edf_chem_name,
       edf_substance_id,

score_chem_name_match_score(upper(edf_chem_name),upper('%benzene%'))
          as match_score
from chemical
where upper(edf_chem_name) like
upper('%benzene%');
order by
score_chem_name_match_score(upper(edf_chem_name),upper('benzene')),

score_chem_name_for_sorting(edf_chem_name)
```

We specify the procedure `score_chem_name_match_score` to take two arguments: one the chemical name from the current row, and one the query string from the user. It returns `0` for an exact match, 1 for a chemical whose name begins with the query string, and 2 in all other cases (remember that this is only used in queries where a LIKE clause ensures that every chemical name at least contains the query string. Once we defined this procedure, we'd be able to call it from a SQL query, the same way that we can call built-in SQL functions such as `upper`.

So is this possible? Yes, in all "enterprise-class" relational database management systems. Historically, each DBMS has had a proprietary language for these *stored procedures*. Starting in 1997, DBMS companies began to put Java byte-code interpreters into the database server. Oracle added Java-in-the-server capability with its 8.1 release in February 1999. If you're looking at old systems such as Scorecard, though, you'll be looking at procedures in Oracle's venerable PL/SQL language:

```
create or replace function
score_chem_name_match_score
 (chem_name IN varchar, query_string IN varchar)
return integer
AS
BEGIN
  IF chem_name = query_string THEN
     return 0;
  ELSIF instr(chem_name,query_string) = 1 THEN
     return 1;
  ELSE
     return 2;
  END IF;
END score_chem_name_match_score;
```

Notice that PL/SQL is a strongly typed language. We say what arguments we expect, whether they are IN or OUT, and what types they must be. We say that `score_chem_name_match_score` will return an integer. We can say that a PL/SQL variable should be of the same type as a column in a table:

```
create or replace function
score_chem_name_for_sorting (chem_name IN
varchar)
return varchar
AS
  stripped_chem_name
chem_hazid_ref.edf_chem_name%TYPE;
BEGIN
  stripped_chem_name :=
ltrim(chem_name,'1234567890-+()[],'' #');
  return stripped_chem_name;
END score_chem_name_for_sorting;
```

The local variable `stripped_chem_name` is going to be the same type as the `edf_chem_name` column in the `chem_hazid_ref` table.

If you are using the Oracle application SQL*Plus to define PL/SQL functions, you have to terminate each definition with a line containing only the character "/". If SQL*Plus reports that there were errors in evaluating your definition, you then have to type "show errors" if you want more explanation. Unless you expect to write perfect code all the time, it can be convenient to leave these SQL*Plus incantations in your .sql files. Here's an example:

```
-- note that we prefix the
incoming arg with v_ to keep it
-- distinguishable from the
database column of the same name
-- this is a common PL/SQL
convention
create or replace function
user_group_name_from_id
(v_group_id IN integer)
return varchar
IS
  -- instead of worrying about how
many characters to
  -- allocate for this local
variable, we just tell
  -- Oracle "make it the same type
```

```
as the group_name
  -- column in the user_groups
table"
  v_group_name
user_groups.group_name%TYPE;
BEGIN
  if v_group_id is null
     then return '';
  end if;
  -- note the usage of INTO below,
which pulls a column
  -- from the table into a local
variable
  select group_name into
v_group_name
   from user_groups
   where group_id = v_group_id;
  return v_group_name;
END;
/
show errors
```

Choosing between PL/SQL and Java

How to choose between PL/SQL and Java? Easy: you don't get to choose. In lots of important places, e.g., triggers, Oracle forces you to specify blocks of PL/SQL. So you have to learn at least the rudiments of PL/SQL. If you're going to build major packages, Java is probably a better long-term choice.

Reference

Next: trees

philg@mit.edu

Add a comment