APEX DEVELOPMENT: WATCH IT LIVE

Bill Holtzman, National Air Traffic Controllers Association

I. Introduction

Application Express crosses several information technology disciplines including Oracle PLSQL and SQL, HTML and Javascript. It can be used at many levels of sophistication, from codeless super-wizards to elaborate hand coding. It can also be used at many levels of development, from database design and construction to UI cosmetics.

As a result of all this versatility, the use of ApEx is in many cases an art form. Developers may learn their methods methodically through their own experiences, but often exposure to others' techniques can accelerate the learning curve. The use of demonstration is a powerful way to share methods. This presentation is entirely demonstration for that reason. An office football pool application will be constructed to illustrate an ApEx design process.

Application Express is well suited for office football pool applications. It requires only a minimal investment of time to get an app up and running, is deployable on the internet and your app is reusable year after year.

For those unfamiliar with betting on football, the "favorite" is the team that is expected to win, the "underdog" is the team expected to lose, and the "line" or "point spread" is the number of point the favorite is expected to win by. To determine which team won the bet, you add the line to the underdog's score.

II. Outline

Database and app design

The database consists of just four tables – pools, users, games and picks. Each pool consists of a group of games, and each pick is unique the game and user. In other words, a user can have only one pick for each game.

The app will function to enable users to create their own accounts and receive their login information via automated email. It will also allow the users to enter their football picks, see others' picks, and monitor the status of the pool in terms of wins and losses. It will provide the administrator with an interface for updating the outcome of each game and will present summary and detailed information concerning the results to all users.

Summary of techniques

The demonstration will cover the following in detail.

- The use of scripts to create tables and objects.
- The use of ApEx automation to create pages for entering and updating data to single tables.
- The use of custom authentication.
- The use of SQL and PLSQL-generated SQL regions.
- The use of conditions within report regions.
- Customization of ApEx-generated pages.
- The use of javascript for both pop-up windows and to make entries into the database.

III. Technical discussion

Database design

Only a few tables are needed for this app, just users, pools, games, and picks. Use the following script to create these tables.

```
CREATE TABLE  "BOWL_USERS"
   (    "ID" NUMBER(10,0),
        "USERNAME" VARCHAR2(20),
        "PW" VARCHAR2(20),
        "SEC_LEV" NUMBER(2),
        "FIRSTNAME" VARCHAR2(20),
        "LASTNAME" VARCHAR2(20),
        "EMAIL" VARCHAR2(50),
        CONSTRAINT "BOWL_USERS_PK" PRIMARY KEY ("ID") ENABLE
   )
/

CREATE TABLE  "BOWL_POOLS"
   (    "ID" NUMBER(10,0),
        "NAME" VARCHAR2(50),
        CONSTRAINT "BOWL_POOLS_PK" PRIMARY KEY ("ID") ENABLE
   )
/

CREATE TABLE  "BOWL_GAMES"
   (    "ID" NUMBER(10,0),
        "POOL_ID" NUMBER(10,0),
        "FAV" VARCHAR2(20),
        "DOG" VARCHAR2(20),
        "BDATE" DATE,
        "LINE" NUMBER(3,1),
        "FAV_SCORE" NUMBER(4,0),
        "DOG_SCORE" NUMBER(4,0),
        CONSTRAINT "BOWL_GAMES_PK" PRIMARY KEY ("ID") ENABLE,
        CONSTRAINT "BOWL_GAMES_FK" FOREIGN KEY ("POOL_ID") REFERENCES  "BOWL_POOLS" ("ID") ENABLE
   )
/

CREATE TABLE  "BOWL_PICKS"
   (    "USER_ID" NUMBER(10,0),
        "GAME_ID" NUMBER(10,0),
        "PICK" NUMBER(1,0),
        CONSTRAINT "BOWL_PICKS_PK" PRIMARY KEY ("USER_ID", "GAME_ID") ENABLE,
        CONSTRAINT "BOWL_PICKS_USER_FK" FOREIGN KEY ("USER_ID") REFERENCES  "BOWL_USERS" ("ID") ENABLE,
CONSTRAINT "BOWL_PICKS_GAME_FK" FOREIGN KEY ("GAME_ID") REFERENCES  "BOWL_GAMES" ("ID") ENABLE
   )
/
```

Create corresponding sequences and triggers for each of these tables and then create an administrative user:

```
INSERT INTO BOWL_USERS (username, pw, sec_lev, firstname, lastname, email) values ('bill', 'password',
2, 'Bill', 'Holtzman', 'skyworker@comcast.net')
```

Creating the app

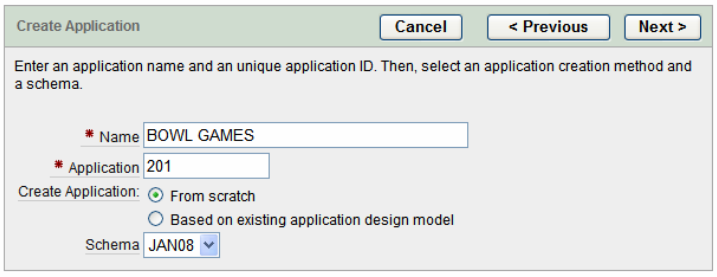


Figure 1. Create application wizard

Create an application from scratch using the wizard as in Figure 1, adding "Report and Form" pages for BOWL_USERS, BOWL_POOLS, and BOWL_GAMES as shown in Figure 2.

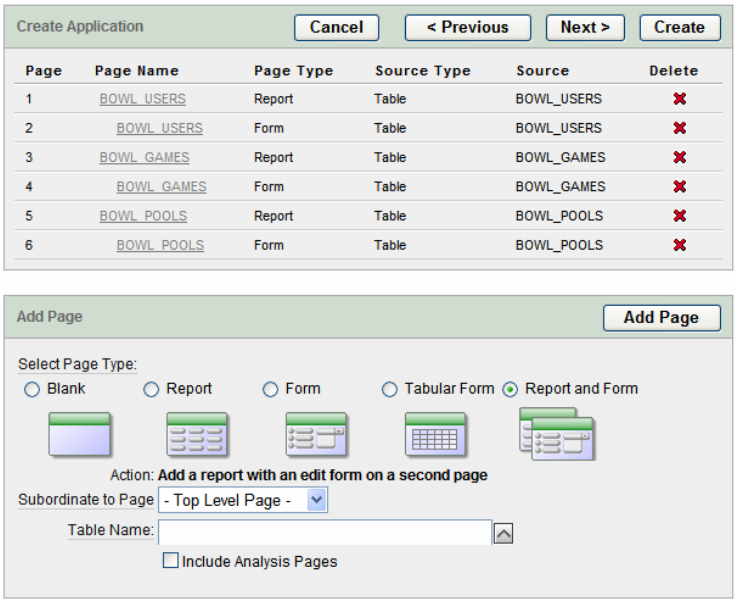


Figure 2. Report and Form wizard

Securing the app

Now create a custom authentication function based on a simple table.

```
CREATE OR REPLACE FUNCTION "BOWL_SECURITY" (p_username in varchar2, p_password in varchar2)
return boolean
  as
  begin
  for c1 in (select 1 from bowl_users where upper(username) = p_username and pw = p_password) loop
  return true;
  end loop;
  return false;
  end;
```

Incorporate the authentication function into the app by creating a new scheme from scratch with the default options except specify your login page as the Invalid Session Target and use a custom function to authenticate as shown in Figure 4.

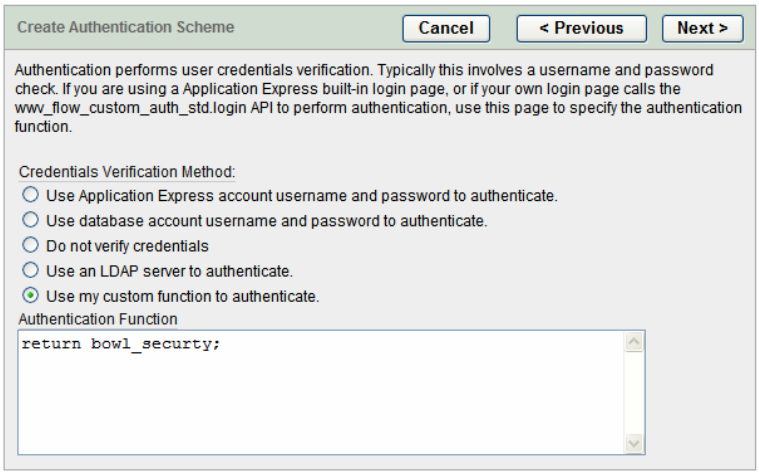


Figure 3. Custom authentication scheme

For the LOGOUT_URL, use:

```
wwv_flow_custom_auth_std.logout?p_this_flow=&FLOW_ID.&p_next_flow_page_sess=&FLOW_ID.
```

Now change your app's current authentication scheme to the new one and then test your scheme by logging in to the app. It should look like Figure 5.

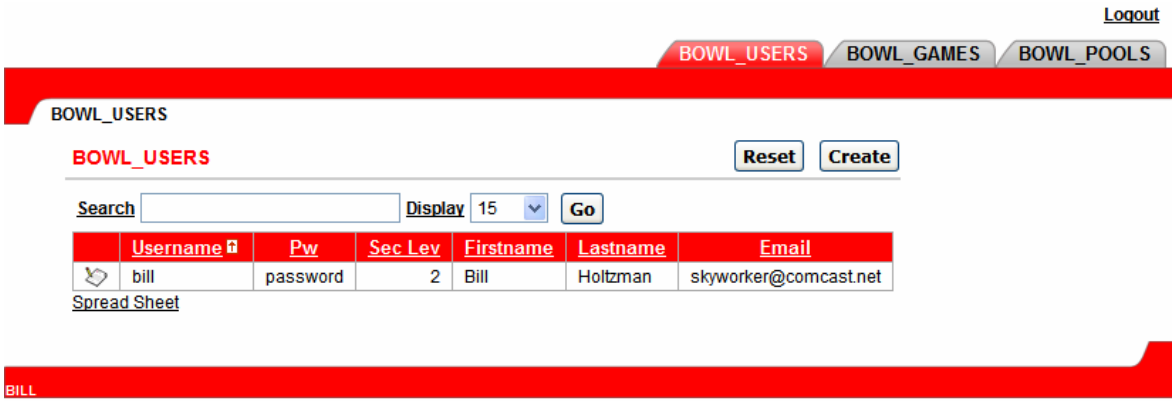


Figure 4. Logged in to the app

Automated account creation

Before we go any farther, I want to set up an automated means for users to create accounts. Create a new page and region. Create three items on that page, P7_FIRSTNAME, P7_LASTNAME, and P7_EMAIL. You can do this with the multiple items feature. Add a button that branches to page 7.

Create a new process to insert a row into BOWL_USERS. Set the process to fire when the Submit button is pressed. You can add an automated email notification to immediately send the user their account credentials.

```
DECLARE
p_pw varchar2(6);
p_username varchar2(100);
BEGIN
insert into bowl_users (firstname, lastname, email, username, pw, sec_lev) values (:P7_FIRSTNAME,
:P7_LASTNAME, :P7_EMAIL, lower(:P7_FIRSTNAME || :P7_LASTNAME), bowl_password(null), 1) returning pw,
username into p_pw, p_username;
apex_mail.send(
   P_TO     =>  :P7_EMAIL,
   P_FROM   =>  'grievance@grievance.natca.net',
   P_SUBJ   =>  'New football pool account',
   P_BODY   =>  'Hi ' || :P7_FIRSTNAME || ',' || chr(10) || 'Here is your new account for the office
football pool. Your username is ' || p_username || ' and your password is ' || p_pw || '. The web site
is located at http://demo.natca.net' || chr(10) || chr(10) || 'Regards,' || chr(10) || 'Bill');
apex_mail.push_queue;
END;
```

The function BOWL_PASSWORD generates a six-character random password with alternating consonants and vowels. On page 7, add the following to the Header section of the page to enable the user to close the new account window when done:

```
<a href="javascript:self.close()">Close Window</a>
```

Now the user could create a new account automatically, if they are able to get to page 7! The page must obviously be public for them to do so. In the security section of the Edit Page screen, ApEx provides a declarative means for doing this in the Edit Page screen.

A means for accessing the page from the login screen is also needed, and this can be done with a javascript pop-up on the login page. Edit page 101 and add the following javascript to the HTML Header:

```
<script language="JavaScript1.2">
function accountPopUp(URL)
  {
day = new Date();
id = day.getTime();
eval("page" + id + " = window.open(URL, '" + id + "',
'toolbar=0,scrollbars=0,location=0,statusbar=0,menubar=0,resizable=0,width=700,height=350,left=150,top=5
0');");
  }
</script>
```

Now add the link HTML to the login page by putting the following code into the Region Footer of the Login Region:

```
<a href="javascript:accountPopUp('f?p=&APP_ID.:7')">Create a new account</a>
```

This link calls the javascript pop-up function and passes in the URL of your New Account page, which you have designated as Public. Try this out by adding a new account.

Customize the automated pages

The automation works well to provide the mechanics of entering and updating each of the tables, but the interface often needs to be modified to accommodate users. This app is intended to manage multiple betting pools. Each pool consists of a group of games. When entering games, the user needs a select list to identify to which pool the game will belong.

On page 4, there is an item P4_POOL_ID. Change this to a Select List, change the label to simply “Pool”, and add the following SQL to the List of values definition:

```
select name d, id r from bowl_pools order by name
```

Create one or more pools using the app and then create a game and the utility of this will become apparent. The Games report region generated by ApEx is shown in Figure 6.

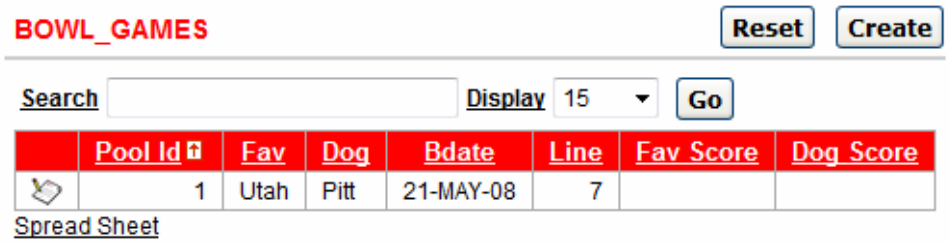


Figure 5. Wizard-generated SQL report region

Change the look to that shown in Figure 7 by changing the Tabular Form Element to “Display as Text (based on LOV, does not save state)” and putting the above query into the List of values definition.

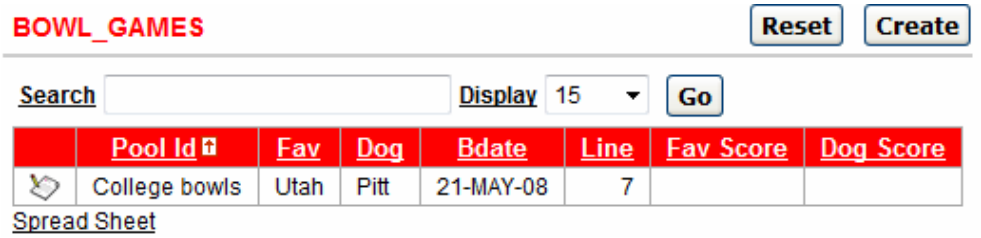


Figure 6. Modified SQL report region

There are scores of other cosmetic and functional changes you can make at this point. The tabs can be renamed for clarity as can the labels. More importantly, the USERS page (page 1) can be either restricted entirely from access by anyone but the admin (SEC_LEV = 2) or at the minimum the passwords and edit buttons can be restricted to admin only. To do the latter, drill down into the Report Attributes page of the BOWL_USERS report and individually add the following exists condition to the ID and PW fields:

```
select 1 from bowl_users where sec_lev = 2 and upper(username) = :APP_USER
```

Creating the "Picks" page

For the user, the core of this application are the pages that allow users to make their picks and see the others' picks, including indicators for wins and losses, who is winning the pool and by how much.

What should this page look like? Here's something like what we want to start with, a listing of the games, dates, and point spreads for a given pool:

Figure 7. Desired initial look

Obviously, the date, game name and line are easily presented using an SQL report region.

```
select to_char(b.bdate, 'Mon FMdd') "DATE", b.fav || ' -' || b.line || ' ' || b.dog "LINE" from
bowl_games b where b.pool_id = :P8_POOL_ID
```

Create page 8 with a SQL report region using this query as the Region Source. Use the existing tab set and create a new tab. Create the P8_POOL_ID item as a select list with submit using the following List of values definition:

```
select name d, id r from bowl_pools order by name
```

You'll also need to create an unconditional branch to page 8. Make sure you have created a pool and entered several games for that pool using the app prior to testing this page to see the results of your work.

Figure 8. First draft of Picks page

So far so good. Now we'd like to see each user's name and their picks in new columns in this SQL report region. This gets a bit trickier because the picks are stored as unique rows in the table BOWL_PICKS, which contains only the fields USER_ID, GAME_ID and PICK. For each user and game there is only one pick. If the value of PICK is 1 then the user has selected the favorite. If PICK is 0 then the user has selected the underdog.

For starters, assume that user BILL has the ID =1 in BOWL_USERS. We can use a subquery to determine BILL's picks for each game:

```
select to_char(b.bdate, 'Mon FMdd') "DATE", b.fav || ' -' || b.line || ' ' || b.dog "LINE", (select
decode(p.pick, 0, substr(b.dog,1,4), 1, substr(b.fav,1,4), 'No pick') from bowl_picks p where p.user_id
= 1 and p.game_id = b.id) "BILL" from bowl_games b where b.pool_id = :P8_POOL_ID
```

Now your report region will look like this:

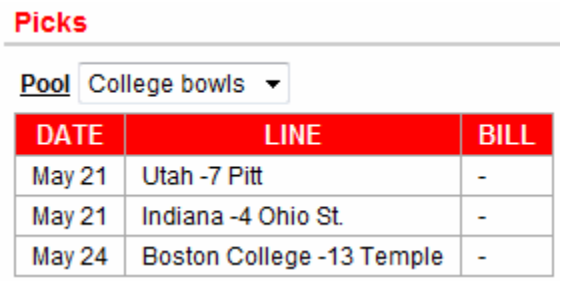


Figure 9. Picks page with hard-coded user column

Of course, we haven't created a mechanism for entering picks yet so there are none! We'll get to that, but first we want to address this issue of manually coding the SQL report Region Source. To give the app the flexibility to add new users without coding, the source SQL must be modified. But how? The only way to enable the report region to automatically add new columns is to construct the SQL on the fly using PLSQL.

To convert our code to PLSQL-generated SQL, first look at this minimal example:

```
declare
p_sql varchar2(32767);
begin
p_sql := q'! select name from bowl_pools !';
return p_sql;
end;
```

The output of this would be:

```
select name from bowl_pools
```

The code uses 10g quoting syntax and generates a SQL statement. What we want is to generate the query that currently defines the report region using PLSQL, substituting in values from the database for the value of p.user_id and for the column name.

```
declare
p_sql varchar2(32767);
cursor c1 is select * from bowl_users order by id;
begin
p_sql := q'! select to_char(b.bdate, 'Mon FMdd') "DATE", b.fav || ' -' || b.line || ' ' || b.dog "LINE"
!';
for a1 in c1 loop
p_sql := p_sql || q'! , (select decode(p.pick, 0, substr(b.dog,1,4), 1, substr(b.fav,1,4), 'No pick')
from bowl_picks p where p.user_id = !' || a1.id || q'! and p.game_id = b.id) "!' || upper(a1.firstname)
|| q'!" !';
end loop;
p_sql := p_sql || q'! from bowl_games b where b.pool_id = :P8_POOL_ID order by bdate desc !';
return p_sql;
end;
```

Change your existing SQL report region PLSQL-generated SQL region with the above source code. You'll need to move P8_POOL_ID to the new region. Make sure you have added more users to your app and then reload the Picks page. It should now have columns for each user.

Figure 10. PLSQL-generated SQL report region

You can view the actual query returned by adding code as shown and then reloading the page:

```
insert into table sql_query (text) values (p_sql);
return p_sql;
end;
```

The following is the result in the above case. (You may notice multiple entries in your text-catching table, and this is due to internal mechanics in Apex.)

```
select to_char(b.bdate, 'Mon FMdd') "DATE", b.fav || ' -' || b.line || ' ' || b.dog "LINE" , (select
decode(p.pick, 0, substr(b.dog,1,4), 1, substr(b.fav,1,4), 'No pick') from bowl_picks p where p.user_id
= 1 and p.game_id = b.id) "BILL" , (select decode(p.pick, 0, substr(b.dog,1,4), 1, substr(b.fav,1,4),
'No pick') from bowl_picks p where p.user_id = 2 and p.game_id = b.id) "BOB" from bowl_games b where
b.pool_id = :P8_POOL_ID order by bdate desc
```

All seems well, but now try to add a new user. You can do this either using the Create button on the Users page or by logging out and using your own New Account page. Now try to view the Picks page. You'll see something like this:

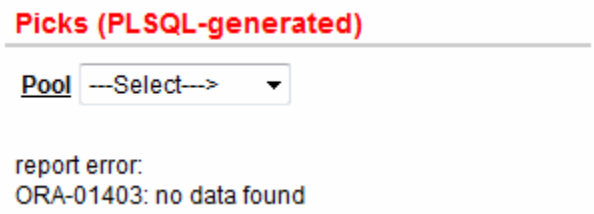


Figure 11. PLSQL-generated SQL report error

The report has a conflict because ApEx has parsed the query and determined the number of columns prior to the addition of a new user. Thus the ApEx engine has to be specifically instructed to re-parse the query every time the report runs. This is done declaratively (thank you Oracle) using the radio button located directly below the Region Source as shown in the figure below. Notice that you can set a limit to the number of columns ApEx will generate using the text field.

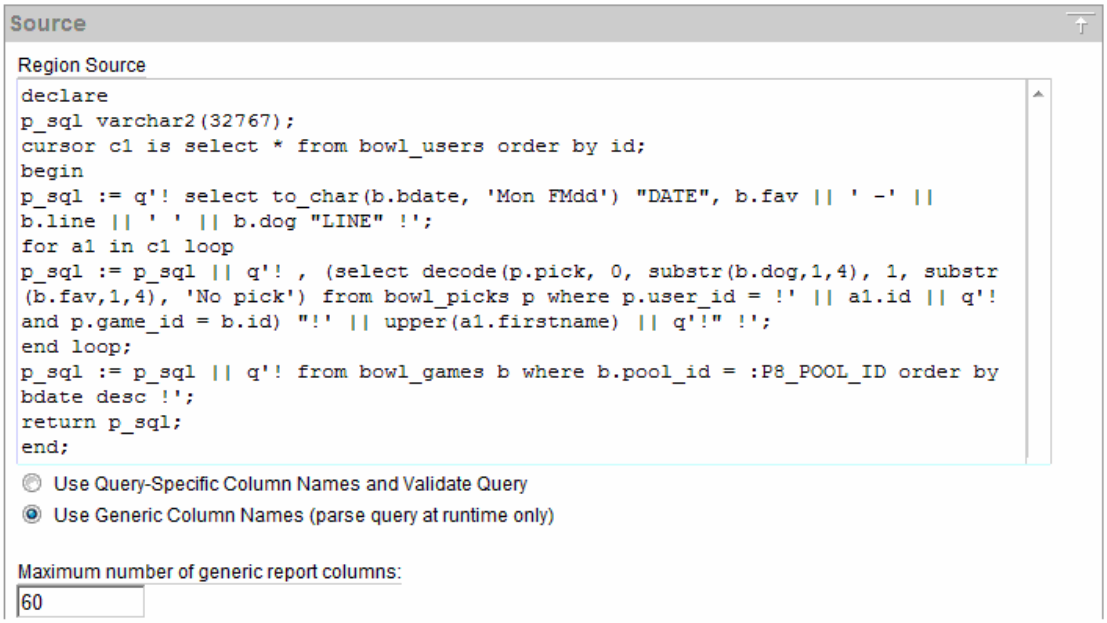


Figure 12. Generic Column Names radio button

So, simply checking the "Generic" radio button solves the problem. If it all were only so easy!

Enabling users to make their picks

It would not be too difficult to create another page for users to enter their picks, but it's easier and more user-friendly to do it on the Picks page. The method here uses javascript to convert each team in the LINE column to a link. That link calls a javascript function that submits the page and fires a process to do the update.

To make each team a link, each team name is wrapped in an HTML link defined as follows:

```
' <a href="javascript$pickEm(''' || b.dog || ''')">' || b.fav || '</a>'
```

The PLSQL is modified as follows with the changes in blue:

```
declare
p_sql varchar2(32767);
cursor c1 is select * from bowl_users order by id;
begin
p_sql := q'! select to_char(b.bdate, 'Mon FMdd') "DATE", '<a href="javascript$pickEm(' || b.id || ',
1)">' || b.fav || '</a> -' || b.line || ' ' || ' <a href="javascript$pickEm(' || b.id || ', 0)">' ||
b.dog || '</a>' "LINE" !';
for a1 in c1 loop
p_sql := p_sql || q'! , (select decode(p.pick, 0, substr(b.dog,1,4), 1, substr(b.fav,1,4), 'No pick')
from bowl_picks p where p.user_id = !' || a1.id || q'! and p.game_id = b.id) "!' || upper(a1.firstname)
|| q'!" !';
end loop;
p_sql := p_sql || q'! from bowl_games b where b.pool_id = :P8_POOL_ID order by bdate desc !';
return replace(p_sql, '$', ': ');
end;
```

Note that a substitution variable must be used within the character string because the presence of a colon would imply a bind variable. The colon is substituted back in when the string is returned. Also note that the same javascript link is used for both the favorite and the underdog, except the value of the second passed variable changes from 1 for the favorite to 0 for the underdog.

The simple javascript function must then be defined. Their only purpose is to pass two values into a page item so that the app knows which record to update (defined by game and user) and the pick (fav or dog).

```
<script language="JavaScript1.2">
  function pickEm(p_game, p_pick)
  {
       document.getElementById("P8_GAME_ID").value = p_game;
       document.getElementById("P8_PICK").value = p_pick;
       doSubmit('pick');
  }
</script>
```

The next step is to create the hidden items specified above (P8_GAME_ID, P8_PICK) and then specify a process to update the user's pick once they click on the team they want to select. It would look like this:

```
declare
p_user_id number(10);
p_count number(1);
begin
select id into p_user_id from bowl_users where upper(username) = :APP_USER;
select count(*) into p_count from bowl_picks where user_id = p_user_id and game_id = :P8_GAME_ID;
if p_count = 0 then
insert into bowl_picks (user_id, game_id, pick) values (p_user_id, :P8_GAME_ID, :P8_PICK);
else
update bowl_picks set pick = :P8_PICK where user_id = p_user_id and game_id = :P8_GAME_ID;
end if;
end;
```

This process fires when Request = pick, so that the above Javascript will trigger it.

Keeping track

The app needs to be able to show the users who is winning, both in summary and detail. The summary can be posted fairly easily with the help of two functions. The first function simply adds up the number of winners each user has for a particular pool.

```
CREATE OR REPLACE FUNCTION  "BOWL_WINS" (p_user_id IN number, p_pool_id IN number)
return number
  IS
  p_pick number(2);
  p_dog_score number(2);
  p_wins number(2) := 0;
  cursor c1 is select * from bowl_games where dog_score is not null and fav_score is not null and
pool_id = p_pool_id;
BEGIN
for a1 in c1 loop
select pick into p_pick from bowl_picks where game_id = a1.id and user_id = p_user_id;
p_dog_score := a1.dog_score + trunc(a1.line);
case
when a1.fav_score > p_dog_score and p_pick = 1 then p_pick := 1;
when a1.fav_score < p_dog_score and p_pick = 0 then p_pick := 1;
else
p_pick := 0;
end case;
p_wins := p_wins + p_pick;
end loop;
return p_wins;
END;
```

The second function creates a text string containing the standings of all users in descending order for posting to the Picks page.

```
CREATE OR REPLACE FUNCTION  "BOWL_RESULTS" (p_pool_id IN number)
return varchar2
  IS
  p_results varchar2(2048) := 'Current wins: ';
  p_wins number(2);
  cursor c1 is select firstname || , 'lastname' “NAME”, bowl_wins(id, pool_id) "WINS" from bowl_users
order by "WINS" desc;
BEGIN
for a1 in c1 loop
p_results := p_results || a1.”NAME” || ' ' || a1."WINS" || ', ';
end loop;
p_results := rtrim(p_results, ', ');
return p_results;
END;
```

To post the results, create a hidden item, P8_RESULTS, on page 8. Set a computation to fire before the header to calculate the value of P8_RESULTS as an SQL query:

```
select bowl_results(:P8_POOL_ID) from dual
```

Then post this string to the page by using the substitution syntax &P8_RESULTS. in the Region Footer of the Picks region. The result is show below.

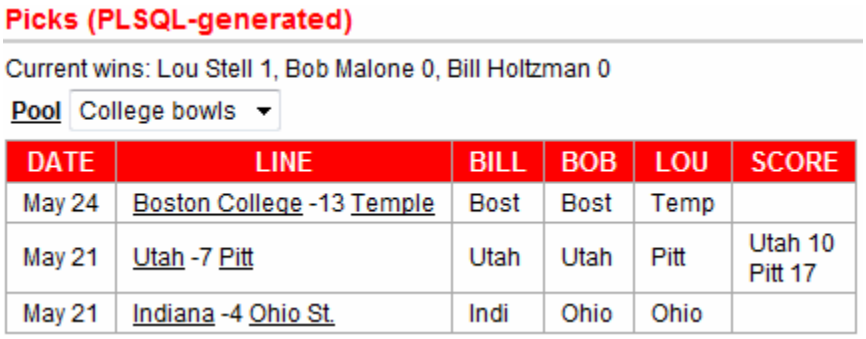
There are two more aspects to keeping track of the pool. First is to present the final score on the Picks page. To do this, we just need to add a column for the score and create a function to format the score.

```
CREATE OR REPLACE FUNCTION  "BOWL_SCORE" (p_id IN number)
return varchar2
  IS
  score varchar2(2048);
  cursor c1 is select * from bowl_games where id = p_id;
BEGIN
for a1 in c1 loop
if a1.fav_score is not null and a1.dog_score is not null then
score := a1.fav || ' ' || a1.fav_score || '<br>' || a1.dog || ' ' || a1.dog_score;
else
score := ' ';
end if;
end loop;
return score;
END;
```

This function gives us the score in the format we want. Now we simply add this column to the PLSQL-generated report, with the new column in blue.

```
declare
p_sql varchar2(32767);
cursor c1 is select * from bowl_users order by id;
begin
p_sql := q'! select to_char(b.bdate, 'Mon FMdd') "DATE", '<a href="javascript$pickEm(' || b.id || ',
1)">' || b.fav || '</a> -' || b.line || ' ' || ' <a href="javascript$pickEm(' || b.id || ', 0)">' ||
b.dog || '</a>' "LINE" !';
for a1 in c1 loop
p_sql := p_sql || q'! , (select decode(p.pick, 0, substr(b.dog,1,4), 1, substr(b.fav,1,4), 'No pick')
from bowl_picks p where p.user_id = !' || a1.id || q'! and p.game_id = b.id) "!' || upper(a1.firstname)
|| q'!" !';
end loop;
p_sql := p_sql || q'! , bowl_score(b.id) "SCORE" from bowl_games b where b.pool_id = :P8_POOL_ID order
by bdate desc !';
return replace(p_sql, '$', ': ');
end;
```

The result of the above changes is shown in Figure 13.



Picks (PLSQL-generated)

Current wins: Lou Stell 1, Bob Malone 0, Bill Holtzman 0

Pool College bowls

DATE	LINE	BILL	BOB	LOU	SCORE
May 24	Boston College -13 Temple	Bost	Bost	Temp	
May 21	Utah -7 Pitt	Utah	Utah	Pitt	Utah 10 Pitt 17
May 21	Indiana -4 Ohio St.	Indi	Ohio	Ohio	

Figure 13. Adding the Score column

One final enhancement gives detailed indications of losers. Create a function that delivers the appropriate strike-through HTML tags and then incorporate this function into the PLSQL that generates the region.

```
CREATE OR REPLACE FUNCTION  "BOWL_WINNER" (p_id IN number)
return number
  IS
  p_winner number(2);
  cursor c1 is select * from bowl_games where id = p_id;
BEGIN
for a1 in c1 loop
if a1.fav_score is not null and a1.dog_score is not null then
p_winner := a1.dog_score + trunc(a1.line);
if a1.fav_score > p_winner then
p_winner := 1;
else
p_winner := 0;
end if;
end if;
end loop;
return p_winner;
END;
```

This function determines if the favorite won (returns 1) or lost (returns 0).

```
CREATE OR REPLACE FUNCTION  "BOWL_STRIKE" (p_gameid IN number, p_userid IN number, p_tag IN number)
return varchar2
  IS
  strike varchar2(30);
  p_pick number(2);
  p_result number(2);
  cursor c1 is select * from bowl_games where id = p_gameid;
BEGIN
for a1 in c1 loop
select pick into p_pick from bowl_picks where userid = p_userid and gameid = p_gameid;
p_result := p_pick + bowl_winner(p_gameid);
if p_result is not null then
case
when p_result = 1 and p_tag = 0 then strike := '<strike>';
when p_result = 1 and p_tag = 1 then strike := '</strike>';
else strike := null;
end case;
end if;
end loop;
return strike;
END;
```

This function returns null unless the user made the wrong pick on the game. Then it either returns <strike> or </strike>, depending on whether the parameter p_tag is 1 or 0. Incorporate this into the Picks region as follows with the new code in blue.

```
DECLARE
p_sql varchar2(32767);
cursor c1 is select * from bowl_users order by id;
BEGIN
p_sql := q'! select to_char(b.bdate, 'Mon FMdd') "DATE", '<a href="javascript$pickEm(' || b.id || ',
1)">' || b.fav || '</a> -' || b.line || ' ' || ' <a href="javascript$pickEm(' || b.id || ', 0)">' ||
b.dog || '</a>' "LINE" !';
for a1 in c1 loop
p_sql := p_sql || q'! , bowl_strike(b.id, !' || a1.id || q'! , 0) || (select decode(p.pick, 0,
substr(b.dog,1,4), 1, substr(b.fav,1,4), 'No pick') from bowl_picks p where p.user_id = !' || a1.id ||
q'! and p.game_id = b.id) || bowl_strike(b.id, !' || a1.id || q'! , 1) "!' || upper(a1.firstname) ||
q'!" !';
end loop;
p_sql := p_sql || q'! , bowl_score(b.id) "SCORE" from bowl_games b where b.pool_id = :P8_POOL_ID order
by bdate desc !';
return replace(p_sql, '$', ': ');
END;
```

The final result is shown below in Figure 14.

Picks (PLSQL-generated)

Current wins: Bill Holtzman 1, Lou Stell 1, Bob Malone 0

Pool College bowls

DATE	LINE	BILL	BOB	LOU	SCORE
May 24	Boston College -13 Temple	Bost	Bost	Temp	
May 21	Utah -7 Pitt	~~Utah~~	~~Utah~~	Pitt	Utah 10 Pitt 17
May 21	Indiana -4 Ohio St.	Indi	~~Ohio~~	~~Ohio~~	Indiana 17 Ohio St. 3

Figure 14. The strike-through feature

IV. Conclusion

This paper detailed exactly how to build an office football pool application. The following methods were presented:

- Use of ApEx automation to add and update records to individual tables.
- Customization of ApEx-generated add and update pages.
- Use of ApEx automation to implement custom authentication.
- Construction of self-service account creation tools with automated email notifications.
- Use of PLSQL-generated SQL report regions with Generic columns to manage the display of additional columns and to provide other enhancements to the report.
- Use of javascript functions within the PLSQL-generated SQL report to update the database.
- Use of conditional HTML formatting within the PLSQL-generated SQL report.

ApEx is a powerful, versatile and convenient tool, and developers with just modest experience in SQL and PLSQL can really get going in a hurry. Skills in Javascript and HTML are a big plus and will help the developer create a very comfortable interface for the user. This paper is intended to give developers the benefit of practical experience with ApEx and give them a leg up in the learning process.

You can contact me at skyworker@comcast.net or 703-403-0139.