BASING A FORMS DATA BLOCK ON A JOIN

INTRODUCTION

Oracle9i Forms Developer offers several ways of defining the data for a block. Simple tables, database views, from clause queries, and stored server-side procedures can all be used as data sources. There is, however, one additional option for a data source which is not extensively discussed in the documentation: the subject of this article—basing a block directly on a join.

Note: The technique described here will work with Forms 6i, Oracle9i Forms and Oracle Forms 10*g*

WHY DO THIS?

Given that there are already several options available for populating a block with data from more than one table, why bother to describe another method? The answer here is, I think, one of simplicity. Although database views offer an excellent and seamless way to view joined data within your blocks, the view has to be created in advance, and this may pose a barrier if the developer who needs the view is not in direct control of the schema. If a particular join is only needed once or infrequently, basing a block directly on a join statement may be a more satisfactory solution than getting approval for a schema change, no matter now trivial that might seem to be.

It is also useful to consider this technique as a simple tuning option to remove database roundtrips caused by the basic use of the Forms Post-Query trigger to retrieve non-base table data. If your code currently does a lookup (or more than one) in the database for every row retrieved in a Post-Query trigger, by changing to using the technique described here you will reduce both your database roundtrips and cursor usage by combining both the base table and the reference lookups into one query operation.

The final benefit of join blocks like this is the fact that you can order the block based on any of the fields in the block, including those fields which might have previously been populated by Post-Query. "How do I order by a non-base table field?" is a common help request. Well, here's how!

PREPARATION

For the sake of simplicity here, we'll use the standard EMP and DEPT demonstration tables in this example. The technique can be extended to any join between two or more tables just as easily. The join condition may be simple, as in this example, or may be complex. It doesn't matter; the principle is the same.

As part of the deal here, one and only one table in the join can be viewed as read-write (we'll call this the master table). The master table will be the recipient of all of the inserts, updates, and deletes (also called the data manipulation language, or DML for short) for the block. Any other tables in the block will only be used to contribute data at query time and have no part to play when DML is being considered. Of course, it's still possible to use transactional triggers within the block to fan DML to more than one table, but that is outside of the scope of this article.

In addition to the names of the tables to join and the join condition itself, you'll need to know the primary key of the master table. You'll need this information because we have to tell Forms what criteria to use when locking the table—but more of that later.

To summarize the information that we need for this simple example:

Tables	Emp Dept
Join Condition	Emp.Deptno = Dept.Deptno
Master Table	Emp
Primary Key of Master Table	Empno

DEFINING THE JOIN BLOCK

The one problem with using this technique of joining directly in the block is that there is no support for the operation from within the Forms Data Block Wizard. To define such a join-based block, we'll have to work directly in the property palette.

Follow these steps:

1) To start off with, as a shortcut, you'll find it simplest to define the master table (EMP in this case) through the Forms Data Block Wizard in the normal way. This does most of the work for you, as generally you'll only be adding a handful of extra fields from the joined table(s). Once the block is defined through the wizard, you can then go through and edit the properties directly in the property palette. Of course, if you are going to be adding a large number of extra fields, you might find it useful to create a separate block for each table that will be part of the join, using the Forms Data Block Wizard in each case, and then carry out a merging exercise to move all the non-master fields into the master block.

2) In the property palette for the block, we need to define the names of the tables that will be contributing to the join. The name of the master table will already be in there if you used the wizard, so just add the detail members of the join to the Query Data Source Name property, using commas to separate the table names (see Figure 1). The list as it appears in the property inspector will be exactly what is submitted in the from clause in the generated SQL that is executed when the block is queried. At this point, if you used the wizard to define the master block information, you might want to rename the block so that it becomes evident that this block is not simply based on the master table.

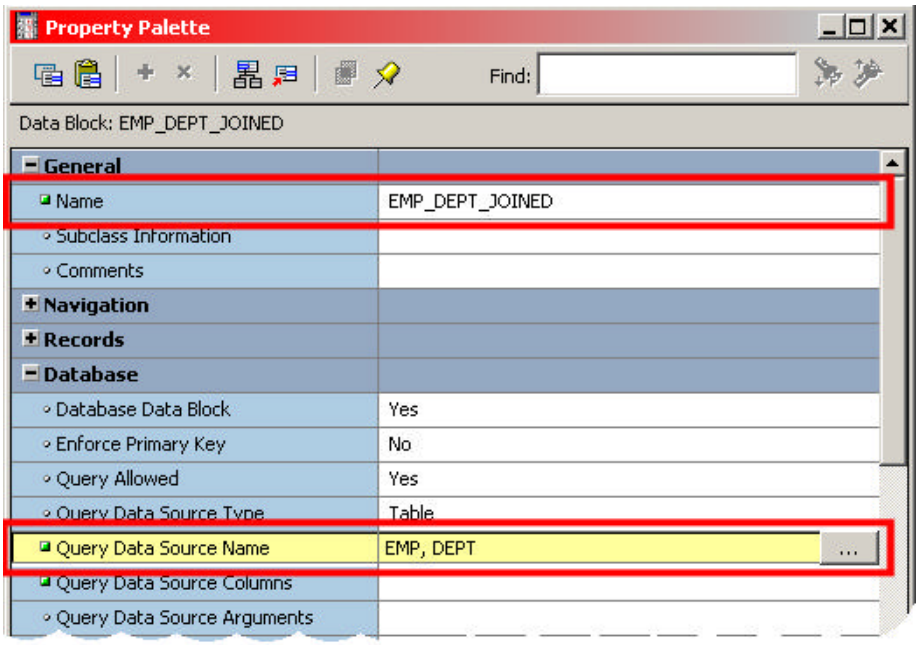

Figure 1 - Define the Query Data Source Name

3) Now that we have defined the list of tables that will make up the join, we'll have to define the join condition as well. To do this, stay in the property palette for the block and enter the join condition into the WHERE Clause property, as shown in Figure 2. Be sure to prefix the column names with the relevant table names, or aliases if you defined the table list with them. Don't use colons to prefix the table names here.

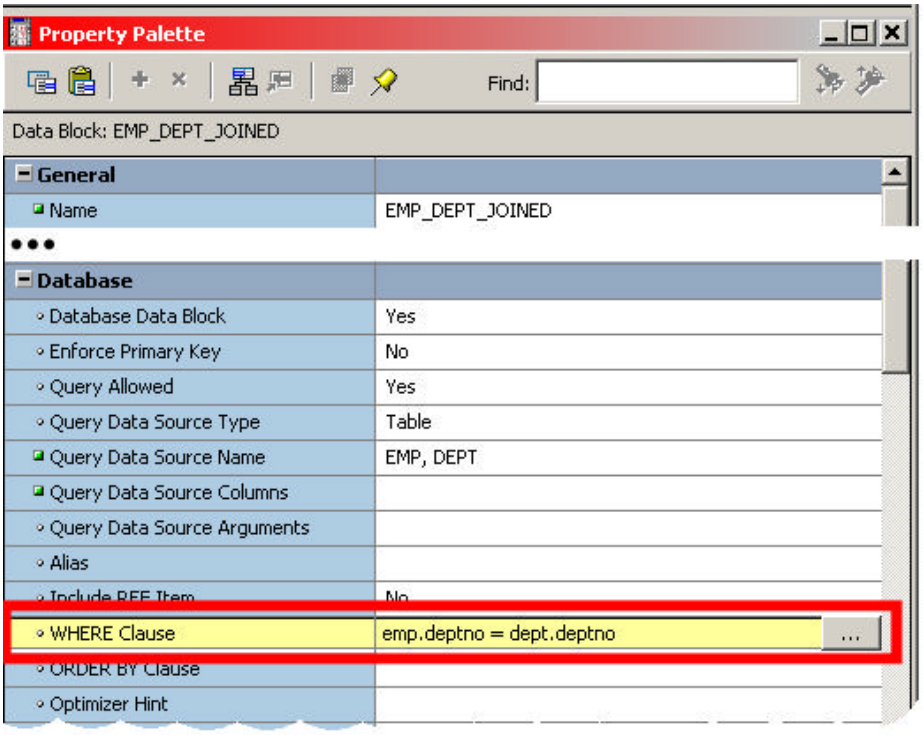

Figure 2 - Add the Join Condition to the WHERE Clause Property

4) So far we've defined what tables will be involved in the join and how they are joined. Next, let's turn our attention to the DML. We have to define the name of the master table in the Advanced Database section of the block property palette. Set the DML Data Target Name property to the name of the master table (see Figure 3). Normally, this is a property that Forms infers the value of based on the base table of the block, but given that there is now a choice of tables we have to indicate explicitly which table should be the recipient of the locks, inserts, updates, and deletes.

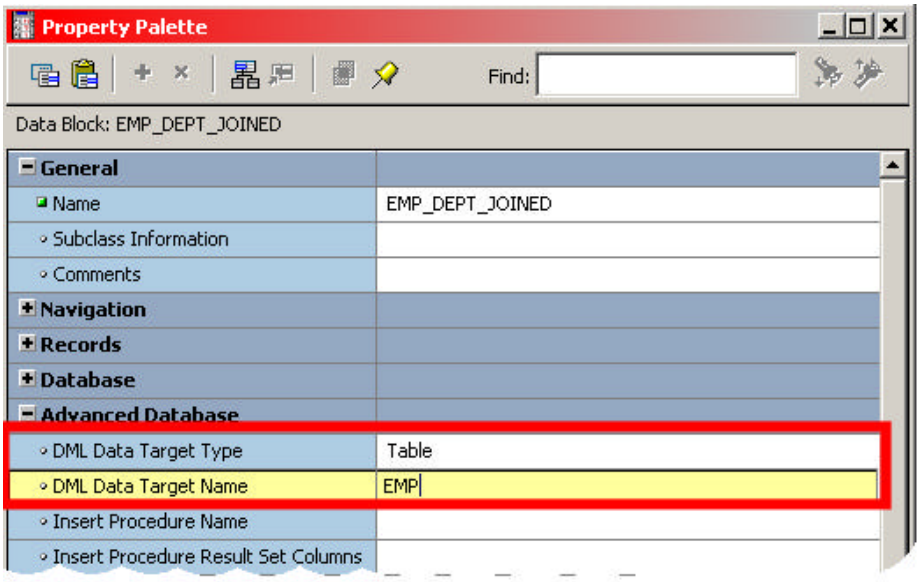

Figure 3 - Define the DML Target for the Block

5) One final thing to do with the master block properties before the joined columns are added is to define the information used to lock the row in the master table. Normally, when a block is based on a single Oracle table, Forms will automatically use the ROWID of the row to handle locking and updates back to that row. Because we are dealing with more than one table here, we'll need to be explicit about how Forms should carry out the lock. This involves setting the Key Mode property on the block to either Non-Updateable (recommended) or Updateable, as shown in Figure 4). This will tell Forms to use the column or columns defined as the primary key for the block as the key to use when locking, updating, or deleting from the master table.

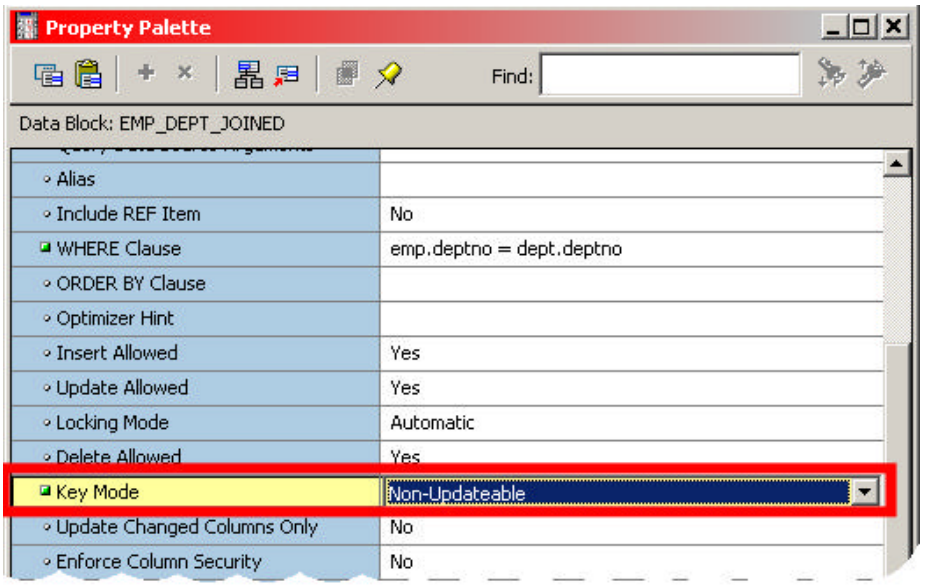

Figure 4 - Set the Key Mode of the Block

To complement the setting of such a key mode, you will also have to define the key column(s) in the block as being the primary key. To do this, bring up the property palette for the key column and set the **Primary Key** property to *Yes* (see Figure 5). If the primary key for the block is a combination of more than one column, set this property for each field.

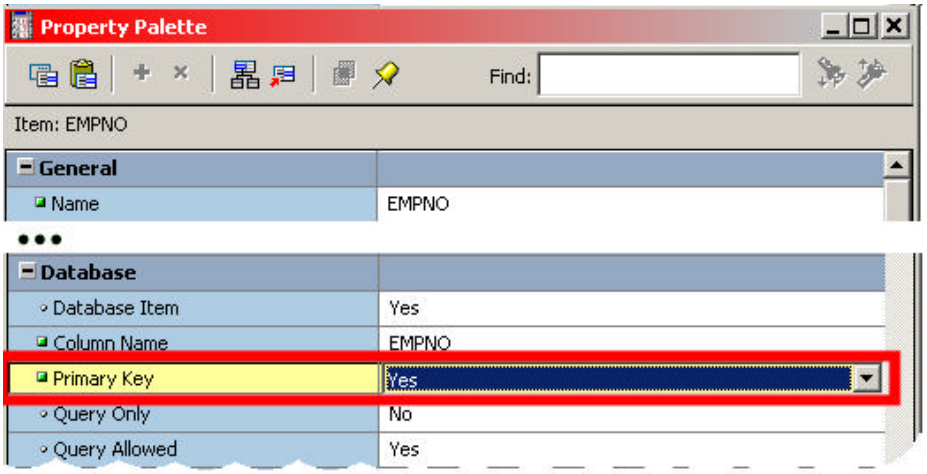

Figure 5 - Define the Primary Key Column

6) Now that all the basics are defined, the extra fields from the join tables can be added to the block. For each field that is added, you'll need to set the following properties, all of which are shown in Figure 6:

Column Name – Set this to the column name that is feeding this field. You'll find it clearer during maintenance if you prefix the column name with the name of the source table, or an alias for it (as defined in the block Query Data Source property). In fact, it is probably wise to prefix the name of each column, including those from the master table, with the relevant table name. If you have columns with the same name in some of the joined tables, you will get an error on query if you've not prefixed in this way, as the column name will be ambiguous. Adopt prefixing as good practice.

Query Only – should be set to *Yes*. These fields from the joined table(s) are read-only.

Insert Allowed – set to *No*.

Update Allowed – set to *No*.

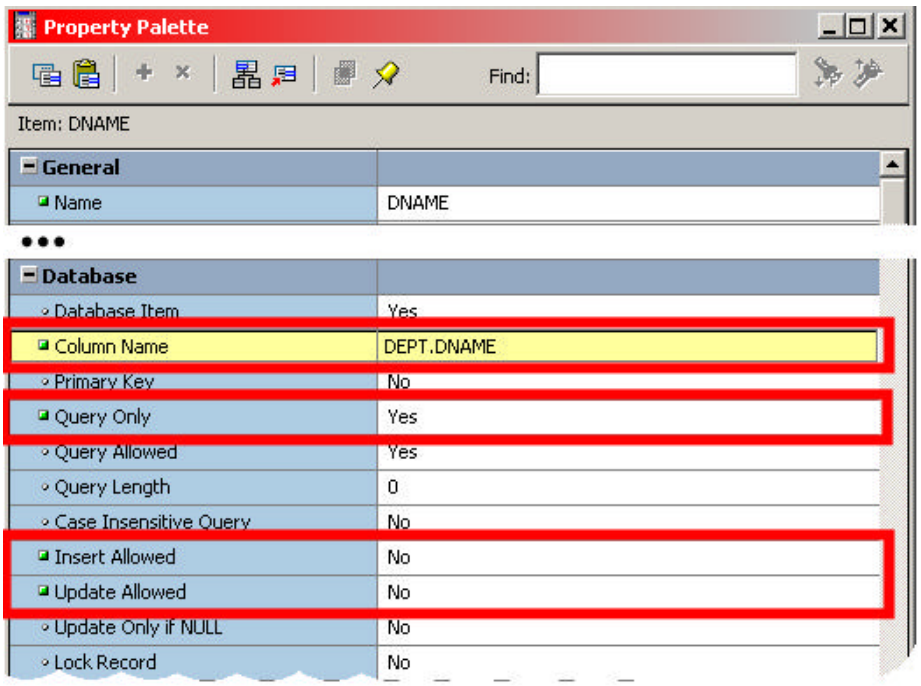

Figure 6 - Define the Properties of Non-Master Columns

Things to Watch Out For

There are several different properties to set in several different places to get this all working, so it is easy to make mistakes and miss something.

Some of the common problems are:

FRM-40505 when executing a query. The query submitted by Forms has not worked for some reason. To diagnose this, press the Display Error key and a further dialog with the actual statement submitted to the database and the underlying error will be presented. The usual cause here is an *ORA-00918: column ambiguously defined* error, which will happen if you have columns with the same name in multiple tables in the join and you've not prefixed the table name or alias into the item Column Name property.

FRM-40501 Oracle Error unable to reserve record for update or delete when changing a record, and pressing Display error gives you an *ORA-00904: invalid column name* error. Check that you've set all of the non-master table columns to query only.

If you are manipulating the where clause of the block at runtime using *Set_Block_Property()*, make sure that you remember to include the basic join condition in any changes you make to the where clause.

CONCLUSION

Basing a block on a direct join is simple to do and provides the benefits of reducing database roundtrips and allowing sorting on non-base table columns, all without having to define special views or stored procedures in the database to do the job.

Whenever you find yourself writing a Post-Query trigger to retrieve reference data into a block, consider using this technique to optimize and simplify your code.