# How Are We to Think and Write Declaratively?

My intention herein is to provide examples of how procedural solutions written in PL/SQL typically have a superior alternative form that is declarative in style. Such declarative solutions:

* Perform better
* Are easier to understand and change
* Often shrink that semantic gap between the implemented program and the specifications which informed the creation of the program in the first place.

When you are not familiar with the power of SQL, and how it can be used to process thousands of records in exotic ways, with a single statement, then you will only use it to perform the very low-level, granular tasks of record retrieval and updating. In other words, SQL becomes a clumsy replacement for COBOL’s OPEN and READ verbs, and each table is a separate indexed file, accessed in dynamic mode.

Using SQL in this manner is somewhat like refusing to use high-level language compilers because you do not believe that the compiler can generate efficient low-level machine code often enough that it just is not worth the time and money to program at a lower level in assembly language. This is most unfortunate, because such an attitude leads to much frustration, on the part of the Oracle application programmer (who, by refusing to use SQL effectively has condemned his application programs to mediocre performance and higher maintenance expense) and the customer/end-user (who does not care about programming styles but who does care software performance, stability, and correct results.)

The Oracle query optimizer does a good job of processing queries, and deriving execution plans, when it has the right statistics to work with. When you try to craft procedural solutions to business problems in Oracle, you are:

* Throwing away the money spent to acquire a relational database, and the powerful tool that is the cost-based optimizer: you are betting that you can come up with a more efficient data access programs than the Oracle cost-based optimizer.
* You are placing successful completion of your project in jeopardy.  
  And even if you manage to drag your procedural solution across the line and force it to work with smallish sets of test data…
* …you next have the hurdle of making your solution scale for real customers, and you take on the huge risk of delivering disappointing performance to them. (Disappointed customers often take their business elsewhere when they get the chance.)

Why take those risks when there are such huge rewards that come from learning to use Oracle’s query optimizer effectively?

# First Example:

Here is the original procedure’s text:

procedure check\_rx\_linkage(target\_facility\_id in varchar) is  
 cursor recs\_to\_check(p\_target\_facility\_id in varchar) is  
 select rowid,  
 dataset\_num,  
 run\_num,  
 rx\_record\_num,  
 rx\_fill\_seq,  
 dur\_conflict\_num,  
 link\_type,  
 target\_facility\_id,  
 target\_rx\_number,  
 target\_fill\_id  
 from RX\_LINKAGE  
 where ( p\_target\_facility\_id = target\_facility\_id   
 or p\_target\_facility\_id is null);  
  
 rx\_linkage\_rec recs\_to\_check%rowtype;  
 type type\_rx\_linkage is table of recs\_to\_check%rowtype;  
 rx\_linkage\_table type\_rx\_linkage;  
 v\_rx\_record\_num RX.rx\_record\_num%type;  
 v\_rx\_fill\_seq RX\_FILL.rx\_fill\_seq%type;  
 v\_facility\_num number(18);  
 type type\_lookup is table of number(18) index by varchar(30);  
 facility\_nums type\_lookup;  
 cursor cur\_facilities(p\_target\_facility\_id in varchar) is  
 select facility\_id,  
 facility\_num  
 from FACILITY  
 where facility\_id is not null  
 and ( p\_target\_facility\_id = facility\_id   
 or p\_target\_facility\_id is null);  
  
 v varchar(10);  
 n number(18);  
  
 del char;  
  
begin  
  
 if(pkg\_facility\_id is not null) then  
 log(log\_debug, 'Checking Rx Linkage');  
 end if;  
  
 open cur\_facilities(target\_facility\_id);  
 loop  
 fetch cur\_facilities  
 into v, n;  
 exit when cur\_facilities%notfound;  
 facility\_nums(v) := n;  
 if(pkg\_facility\_id is not null) then  
 log(log\_debug, 'Facility ' || v || ' = ' || n);  
 end if;  
 end loop;  
 close cur\_facilities;  
  
 open recs\_to\_check(target\_facility\_id);  
 loop  
 fetch recs\_to\_check bulk collect  
 into rx\_linkage\_table limit 1000;  
 exit when rx\_linkage\_table.first is null;  
  
 del := 'N';  
  
 if(pkg\_facility\_id is not null) then  
 log(log\_debug, 'Batch of ' || sql%rowcount);  
 end if;  
  
 for i in rx\_linkage\_table.first .. rx\_linkage\_table.last loop  
 rx\_linkage\_rec := rx\_linkage\_table(i);  
 v\_facility\_num := facility\_nums(rx\_linkage\_rec.target\_facility\_id);  
  
 if(pkg\_facility\_id is not null) then  
 log(log\_debug, 'Checking Linkage ' ||   
 rx\_linkage\_rec.rx\_record\_num);  
 end if;  
  
 begin  
 -- I have come to the conclusion that this query is wasted effort,  
 -- when  
 -- rx\_linkage\_rec.link\_type != 'R'  
 select max(rx\_record\_num)  
 into v\_rx\_record\_num  
 from RX  
 where ( facility\_num = v\_facility\_num )  
 and ( rx\_number = rx\_linkage\_rec.target\_rx\_number );  
  
 if rx\_linkage\_rec.target\_fill\_id is not null then  
 select rf.rx\_record\_num, rf.rx\_fill\_seq  
 into v\_rx\_record\_num, v\_rx\_fill\_seq  
 from RX\_FILL\_XREF rfx  
 join RX\_FILL rf on rf.rx\_fill\_seq = rfx.internal\_rx\_fill\_id  
 and rf.rx\_record\_num = rfx.internal\_rx\_id  
 where rfx.facility\_num = v\_facility\_num  
 and rfx.rx\_number = rx\_linkage\_rec.target\_rx\_number  
 and rfx.external\_fill\_id = rx\_linkage\_rec.target\_fill\_id;  
 else  
 v\_rx\_fill\_seq := null;  
 end if;  
  
 if(pkg\_facility\_id is not null) then  
 log(log\_debug, 'Found Target Rx');  
 end if;  
  
 if rx\_linkage\_rec.link\_type = 'R' then  
 update RX  
 set reassigned\_rx\_num = v\_rx\_record\_num  
 --num\_updates = num\_updates + 1,  
 --sys\_user = my\_user(),  
 --datestamp = sysdate  
 where ( rx\_record\_num = rx\_linkage\_rec.rx\_record\_num );  
  
 if(sql%rowcount > 0) then  
 del := 'Y';  
 end if;  
 end if;  
  
 if rx\_linkage\_rec.link\_type = 'L' or  
 rx\_linkage\_rec.link\_type = 'D' then  
 if rx\_linkage\_rec.link\_type = 'L' then  
 update DUR\_CONFLICT\_DUPL\_DRUG  
 set rx\_record\_num = v\_rx\_record\_num,  
 facility\_num = v\_facility\_num,  
 rx\_fill\_seq = v\_rx\_fill\_seq  
 --num\_updates = num\_updates + 1,  
 --sys\_user = my\_user(),  
 --datestamp = sysdate  
 where dur\_conflict\_num = rx\_linkage\_rec.dur\_conflict\_num;  
 elsif rx\_linkage\_rec.link\_type = 'D' then  
 update DUR\_CONFLICT\_DRUG\_DRUG  
 set rx\_record\_num = v\_rx\_record\_num,  
 facility\_num = v\_facility\_num,  
 rx\_fill\_seq = v\_rx\_fill\_seq  
 --num\_updates = num\_updates + 1,  
 --sys\_user = my\_user(),  
 --datestamp = sysdate  
 where dur\_conflict\_num = rx\_linkage\_rec.dur\_conflict\_num;  
 end if;  
  
 if(sql%rowcount > 0) then  
 del := 'Y';  
 end if;  
 end if;  
  
 if(del = 'Y') then  
 delete from RX\_LINKAGE  
 where rowid = rx\_linkage\_rec.rowid;  
 end if;  
  
 exception when NO\_DATA\_FOUND then  
  
 if(pkg\_facility\_id is not null) then  
 log(log\_debug, 'No Data Found: ' ||   
 rx\_linkage\_rec.rx\_record\_num);  
 end if;  
  
 end;  
  
 end loop;  
  
 if(pkg\_facility\_id is not null) then  
 log(log\_debug, 'Commit');  
 end if;  
  
 commit;  
  
 end loop;  
 close recs\_to\_check;  
  
end;

The following shows that it is possible to write a declarative version of the program that will vastly preferable to its procedural counterpart. It will suffer from fewer performance problems, and because it is easier to understand what the program does, it will be easier to see the connection between the program and its business purpose.

Points to consider:

* The facility table will be touched or used by the three different statements in the new version of the procedure. The original procedure loads it once into a collection, and then a main processing loop procedurally probes the collection, based on data fetched from other tables (via other cursors and queries.) On the surface, this seems like it would perform much better than trying to use the facility table in three different SQL statements, for it only gets fetched one time, right? However, Oracle can and does cache the data from such small tables, significantly reducing the cost of access. Code written in this fashion begs the question, “Do you really think you can come up with an ad-hoc caching approach that will be, most of the time, far superior to Oracle’s caching?” It is quite possible that the original approach, in using a main processing loop which perfoms queries once per loop iteration, will perform far worse than the declarative solution, because of the overhead of repeatedly invoking the queries in the body of the loop, such as that query which calculates a max(rx\_record\_num) value. At the least, there will be at least one soft parse of that query in each loop iteration.
* The procedural nature of the original solution, by its very structure, makes it very hard to see that the aggregation query (which calculates the max(rx\_record\_num) value) is a wasted call whenever the link\_type of the retrieved rx\_linkage row (retrieved via the recs\_to\_check cursor) is 'R'.
* The new version of the procedure is “flat” in structure, and it is clear that, ultimately, three different tables will be dealt with, and three different kinds of deletions will need to be executed against the rx\_linkage table.
* Using SQL queries, and using the power of SQL to join tables together, takes advantage of the relational database’s ability to match records together, to automatically perform what used to be called “joint file processing.” PL/SQL programs which seek to perform join file processing instead of using join queries will be harder to understand and more expensive to maintain, and will perform worse than equivalent programs which take advantage of join queries.

Here is the text of the new version of the procedure:

procedure check\_rx\_linkage(target\_facility\_id in varchar)  
is  
begin  
  update rx rxo  
  set rxo.reassigned\_rx\_num =  
    (  
      select  
        max(rx.rx\_record\_num)  
          over (partition by rx.facility\_num, rx.rx\_number)  
        max\_rx\_record\_num  
      from  
        rx\_linkage rxl  
      , facility fcty  
      , rx  
      where  
          rxl.target\_rx\_number = rx.rx\_number  
      and fcty.facility\_num    = rx.facility\_num  
      --  
      and (    rxl.target\_facility\_id = target\_facility\_id  
           and fcty.facility\_id = target\_facility\_id  
           or target\_facility\_id is null                   )  
      and rxl.target\_facility\_id = fcty.facility\_id  
      and rxl.link\_type = 'R'  
      --  
      and rxl.rx\_record\_num = rxo.rx\_record\_num  
    )  
  where  
      rxo.rx\_record\_num in (  
        select  
          rxl.rx\_record\_num  
        from  
          rx\_linkage rxl  
        where  
            rxl.link\_type = 'R'  
        and ( rxl.target\_facility\_id = target\_facility\_id   
 or target\_facility\_id is null)  
      )  
  ;  
  --  
  if sql%rowcount != 0 then  
    delete rx\_linkage rxl  
    where rxl.link\_type = 'R'  
      and ( rxl.target\_facility\_id = target\_facility\_id  
 or target\_facility\_id is null)  
    ;  
  end if;  
  --  
  update dur\_conflict\_dupl\_drug dupl\_drug  
  set  
    (  
      dupl\_drug.rx\_record\_num  
    , dupl\_drug.facility\_num  
    , dupl\_drug.rx\_fill\_seq  
    ) =   
    (  
      select  
        rf.rx\_record\_num  f\_rx\_record\_num  
      , rf.rx\_fill\_seq    f\_rx\_fill\_seq  
      , fcty.facility\_num  
      from  
        rx\_linkage rxl  
      , facility fcty  
      , rx\_fill\_xref rfx  
      , rx\_fill rf  
      where  
          (    rxl.target\_facility\_id = target\_facility\_id  
           and fcty.facility\_id = target\_facility\_id  
           or target\_facility\_id is null                   )  
      and rxl.target\_facility\_id = fcty.facility\_id  
      and rxl.link\_type = 'L'  
      --  
      and rf.rx\_fill\_seq = rfx.internal\_rx\_fill\_id  
      and rf.rx\_record\_num = rfx.internal\_rx\_id  
      --  
      and rxl.target\_rx\_number = rfx.rx\_number  
      and fcty.facility\_num = rfx.facility\_num  
      and rxl.target\_fill\_id = rfx.external\_fill\_id  
      --  
      and rxl.dur\_conflict\_num = dupl\_drug.dur\_conflict\_num  
    )  
  where  
      dupl\_drug.dur\_conflict\_num in (  
        select  
          rxl.dur\_conflict\_num  
        from  
          rx\_linkage rxl  
        where  
            link\_type = 'L'  
        and ( rxl.target\_facility\_id = target\_facility\_id   
 or target\_facility\_id is null)  
      )  
  ;  
  --  
  if sql%rowcount != 0 then  
    delete rx\_linkage rxl  
    where rxl.link\_type = 'L'  
      and ( rxl.target\_facility\_id = target\_facility\_id  
 or target\_facility\_id is null)  
    ;  
  end if;  
  --  
  update dur\_conflict\_drug\_drug drug\_drug  
  set  
    (  
      drug\_drug.rx\_record\_num  
    , drug\_drug.facility\_num  
    , drug\_drug.rx\_fill\_seq  
    ) =   
    (  
      select  
        rf.rx\_record\_num  f\_rx\_record\_num  
      , rf.rx\_fill\_seq    f\_rx\_fill\_seq  
      , fcty.facility\_num  
      from  
        rx\_linkage rxl  
      , facility fcty  
      , rx\_fill\_xref rfx  
      , rx\_fill rf  
      where  
          (    rxl.target\_facility\_id = target\_facility\_id  
           and fcty.facility\_id = target\_facility\_id  
           or target\_facility\_id is null                   )  
      and rxl.target\_facility\_id = fcty.facility\_id  
      and rxl.link\_type = 'D'  
      --  
      and rf.rx\_fill\_seq = rfx.internal\_rx\_fill\_id  
      and rf.rx\_record\_num = rfx.internal\_rx\_id  
      --  
      and rxl.target\_rx\_number = rfx.rx\_number  
      and fcty.facility\_num = rfx.facility\_num  
      and rxl.target\_fill\_id = rfx.external\_fill\_id  
      --  
      and rxl.dur\_conflict\_num = drug\_drug.dur\_conflict\_num  
    )  
  where  
      drug\_drug.dur\_conflict\_num in (  
        select  
          dur\_conflict\_num  
        from  
          rx\_linkage rxl  
        where  
            link\_type = 'D'  
        and ( rxl.target\_facility\_id = target\_facility\_id  
 or target\_facility\_id is null)  
      )  
  ;  
  --  
  if sql%rowcount != 0 then  
    delete rx\_linkage rxl  
    where rxl.link\_type = 'D'  
      and ( rxl.target\_facility\_id = target\_facility\_id  
 or target\_facility\_id is null)  
    ;  
  end if;  
  --  
end;

# Second Example:

This is the original procedure:

procedure check\_prescriber\_addresses is  
 type type\_pres\_addr is table of number(18);  
 pres\_addr\_table type\_pres\_addr;  
 curr\_prescriber\_num number(18) := null;  
 primary\_address\_num number(18) := null;  
  
 cursor check\_pres\_addr\_curs (p\_dataset\_num number) is  
 select p.prescriber\_num  
 from prescriber\_base p, prescriber\_xref x  
 where x.dataset\_num = p\_dataset\_num  
 and p.prescriber\_num = x.internal\_id  
 and p.status = 'A'  
 and (select count(\*)  
 from prescriber\_address  
 where prescriber\_num = p.prescriber\_num  
 and status = 'A'  
 and is\_default = 'Y') > 1;  
  
 -- Retrieves all address records for a prescriber that are marked as primary  
 -- and selects one of them to be the primary.  
 function find\_primary\_address (i\_prescriber\_num in number) return number is  
 cursor default\_curs (p\_prescriber\_num number) is  
 select prescriber\_address\_num,  
 datestamp,  
 sys\_user  
 from prescriber\_address  
 where prescriber\_num = p\_prescriber\_num  
 and status = 'A'  
 and is\_default = 'Y'  
 order by datestamp desc;  
  
 default\_rec default\_curs%rowtype;  
 curr\_default\_rec default\_curs%rowtype;  
  
 begin  
 open default\_curs(i\_prescriber\_num);  
 loop  
 fetch default\_curs  
 into default\_rec;  
 exit when default\_curs%notfound;  
  
 -- The first choice for the default prescriber address record will  
 -- be the one with the most recent datestamp that has been modified  
 -- by an actual user. If none has been modified by a user, use the  
 -- one with the most recent datestamp.  
 if (curr\_default\_rec.prescriber\_address\_num is null) then  
 curr\_default\_rec := default\_rec;  
 else  
 if (nvl(curr\_default\_rec.sys\_user, 2) = 2 and  
 default\_rec.sys\_user > 2) then  
 curr\_default\_rec := default\_rec;  
 end if;  
 end if;  
  
 end loop;  
 close default\_curs;  
  
 if (curr\_default\_rec.prescriber\_address\_num is not null) then  
 return curr\_default\_rec.prescriber\_address\_num;  
 end if;  
  
 return null;  
 end find\_primary\_address;  
  
  
begin  
 log(log\_debug, 'Begin check\_prescriber\_address');  
  
 open check\_pres\_addr\_curs(my\_dataset\_num());  
 loop  
 fetch check\_pres\_addr\_curs bulk collect  
 into pres\_addr\_table limit 1000;  
 exit when pres\_addr\_table.first is null;  
  
 for i in pres\_addr\_table.first .. pres\_addr\_table.last loop  
 curr\_prescriber\_num := pres\_addr\_table(i);  
 primary\_address\_num := find\_primary\_address(curr\_prescriber\_num);  
  
 if (primary\_address\_num is not null) then  
 -- update the non-primary addresses  
 update prescriber\_address  
 set is\_default = 'N',  
 sys\_user = 2,  
 num\_updates = num\_updates + 1,  
 datestamp = sysdate  
 where prescriber\_num = curr\_prescriber\_num  
 and prescriber\_address\_num <> primary\_address\_num  
 and status = 'A'  
 and is\_default = 'Y';  
 end if;  
 end loop;  
 commit;  
 end loop;  
 close check\_pres\_addr\_curs;  
  
end check\_prescriber\_addresses;

The following new procedure is already in production use. The original motivation to change the procedure came during performance reviews of the conversion cycle; the statement in the original procedure which caught attention during the performance review was:

select p.prescriber\_num  
 from prescriber\_base p, prescriber\_xref x  
 where x.dataset\_num = p\_dataset\_num  
 and p.prescriber\_num = x.internal\_id  
 and p.status = 'A'  
 and (select count(\*)  
 from prescriber\_address  
 where prescriber\_num = p.prescriber\_num  
 and status = 'A'  
 and is\_default = 'Y') > 1;

The session trace data collected during performance testing showed that this statement was being invoked thousands of times during a conversion cycle. On average, it was causing Oracle to perform, 9,200 i/o operations per invocation. As it turned out, the appropriate solution is to design the procedure declaratively, so that the i/o needed is performed once, and the information which would be retrieved by such a statement is used only once.

This is the proposed new procedure:

PROCEDURE check\_prescriber\_addresses(  
 p\_dataset\_num NUMBER  
 , p\_startrec NUMBER  
 , p\_endrec NUMBER  
)  
IS  
BEGIN  
 -- ERX-27498  
 -- This procedure, plus changes to the way it is invoked, versus its  
 -- former version, should yield superior performance.  
 -- Optimization is achieved primarily through proper use of SQL  
 -- to perform what had been done procedurally.  
 -- This procedure is designed to be included within source file  
 -- dataloader\_prescriber\_body, as a sub-sub-program within the  
 -- dataloader.migrate\_prescriber\_block procedure.  
 UPDATE /\* SQL:CAEAE754-FCA2-56AE-8848-5392AE526507[3000] \*/   
 prescriber\_address pa1  
 SET pa1.is\_default = 'N'  
 , pa1.sys\_user = 2  
 , pa1.num\_updates = pa1.num\_updates + 1  
 , pa1.datestamp = sysdate  
 WHERE  
 pa1.prescriber\_address\_num IN (  
 SELECT  
 u.pa\_prescriber\_address\_num  
 FROM  
 (  
 SELECT  
 p.prescriber\_num  
 , count(\*) OVER (PARTITION BY p.prescriber\_num, x.dataset\_num) pa\_count  
 , pa.sys\_user pa\_sys\_user  
 , pa.prescriber\_address\_num pa\_prescriber\_address\_num  
 , x.dataset\_num  
 --  
 -- If there is at least one prescriber\_address row, for the  
 -- prescriber\_num and dataset\_num, such that the sys\_user was a "real  
 -- user", then this will 1. Otherwise it is 0. The outer query depends   
 -- on this.  
 , max(pa.user\_class)  
 OVER (PARTITION BY p.prescriber\_num, x.dataset\_num)   
 actual\_user\_exists  
 --  
 , pa.datestamp  
 --  
 , last\_value(pa.prescriber\_address\_num)  
 OVER (PARTITION BY p.prescriber\_num, x.dataset\_num, pa.user\_class  
 ORDER BY pa.datestamp, pa.prescriber\_address\_num  
 ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING)  
 last\_prescriber\_address\_num  
 -- The following is used to distinguish between those   
 -- prescriber\_address rows edited by a non-conversion user,   
 -- which is identified by "1", and those edited by a conversion   
 -- user, identified by "0".  
 , pa.user\_class  
 FROM  
 prescriber\_base p  
 , (  
 SELECT  
 pa1.\*  
 , CASE WHEN pa1.sys\_user > 2 THEN 1 ELSE 0 END user\_class  
 FROM  
 prescriber\_address pa1  
 ) pa  
 , (  
 SELECT DISTINCT -- Select distinct used here because it is   
 px.internal\_id -- possible for there to be multiple rows in   
 , px.dataset\_num -- the prescriber\_xref per internal\_key   
 FROM -- value. This inline view provides   
 prescriber\_xref px -- the set of prescriber\_num's, per   
 , stage\_prescriber sp -- dataset\_num, such that the prescriber\_num   
 -- was somehow affected in processing the   
 -- dataset\_num.  
 WHERE  
 sp.dataset\_num = px.dataset\_num  
 AND sp.id = px.external\_id  
 AND p\_startrec <= sp.element\_num  
 AND sp.element\_num <= p\_endrec  
 ) x  
 WHERE  
 p.prescriber\_num = x.internal\_id  
 AND x.dataset\_num = p\_dataset\_num  
 --  
 AND p.status = 'A'  
 AND pa.prescriber\_num = p.prescriber\_num  
 AND pa.status = 'A'  
 AND pa.is\_default = 'Y'  
 ) u  
 WHERE  
 -- This following means that we are only interested in those  
 -- prescriber\_address rows such that there are multiple  
 -- prescriber\_address rows per prescriber:  
 u.pa\_count > 1  
 AND (  
 -- The prescriber\_address row does not have the latest datestamp out of   
 -- all the rows for the same dataset, prescriber, and "user class"   
 -- (either non-conversion user, or conversion user)...  
 u.pa\_prescriber\_address\_num != u.last\_prescriber\_address\_num  
 -- ...or  
 -- There are "actual users" so the only rows that should be set to   
 -- "non-primary" are those for the "conversion user," identified by   
 -- user\_class = 0...  
 or u.actual\_user\_exists = 1 AND u.user\_class = 0  
 -- ...or BOTH disjuncts are satisfied, which means that there were   
 -- rows updated by "actual users," so that we will always update such   
 -- prescriber\_address rows whenever they are not that "last   
 -- prescriber\_address row" AND we will update a row when it is the row   
 -- for the "last prescriber\_address" and it is a row which was updated by   
 -- a "not-actual user".  
 )  
 );  
END  
;