# Array Interface in PL/SQL

Christian Antognini
Trivadis AG
Zürich, Switzerland

## Introduction

The PL/SQL package DBMS_SQL since version 8.0 supports an array interface.
With this new feature it's possible to fetch multiple rows from a database table into a PL/SQL index-by table in one call to the server. Conversely, you can insert all rows from an index-by table into a database table in one call.
The goal of this article is to compare the performance of an array interface processing over database link with a simple one.
The tests were done on version 8.0.5 for Solaris.

## The Job

A local table must be filled with some data that reside on a remote database. The remote and local tables are different. Thus the rows must be converted via a "complex" algorithm to be supported in the local schema (i.e. the conversion cannot be done directly in SQL but a PL/SQL procedure is required). Therefore a simple replication via snapshots, "insert as select" or SQL*Plus copy command cannot be used.
For the tests 2000 rows (the average row size is 100 bytes) are copied from a remote database into a local one. The network bandwidth between the two databases is about of 2 Mbits/s.

### The Tables

Contrary to the job conditions for the tests two tables having the same columns are used (the article goal is attained without unnecessary complications). Here is the definition.

```
Name          Null?    Type
------------ -------- -------------
C1            NOT NULL NUMBER(15)
C2            NOT NULL VARCHAR2(30)
C3            NOT NULL DATE
C4            NOT NULL CHAR(1)
C5            NOT NULL NUMBER(2)
C6                     VARCHAR2(100)
```

### The Database Link

To connect the remote database a synonym and a database link was created.

```
create database link <dblink>
  connect to <user> identified by <password> using '<dbalias>';
create synonym remotetab for remotetab@<dblink>;
```

### The First Test

A simple procedure to make the job is the following.

```
CREATE OR REPLACE PROCEDURE test1 IS
  CURSOR c_rt IS (select * from remotetab);
BEGIN
  FOR r IN c_rt
  LOOP
    -- here data conversion
    INSERT INTO localtab VALUES(r.c1, r.c2, r.c3, r.c4, r.c5, r.c6);
  END LOOP;
END;
```

### The Second Test

In this test to improve the performance the array interface is used to select the records from the remote database and insert they in the local one. The 2000 rows are selected in a single operation (p_numrows = 2000), stored in the UGA memory and also inserted in a single operation. The sample code is the following.

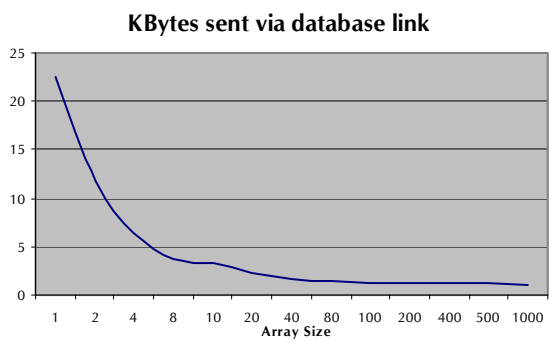
```
CREATE OR REPLACE PROCEDURE test2 (p_numrows NUMBER) IS
  CURSOR    c_rt IS (select * from remotetab);
  l_c1      dbms_sql.number_table;
  l_c2      dbms_sql.varchar2_table;
  l_c3      dbms_sql.date_table;
  l_c4      dbms_sql.varchar2_table;
  l_c5      dbms_sql.number_table;
  l_c6      dbms_sql.varchar2_table;
  l_dummy   INTEGER;
  l_numrows INTEGER;
  c_select  INTEGER;
  l_select  VARCHAR2(100) := 'SELECT * FROM remotetab';
  c_insert  INTEGER;
  l_insert  VARCHAR2(100) := 'INSERT INTO localtab '||
                             'VALUES(:c1, :c2, :c3, :c4, :c5, :c6)';
BEGIN
  c_select := dbms_sql.open_cursor;
  dbms_sql.parse(c_select, l_select, dbms_sql.native);
  dbms_sql.define_array(c_select, 1, l_c1, p_numrows, 1);
  dbms_sql.define_array(c_select, 2, l_c2, p_numrows, 1);
  dbms_sql.define_array(c_select, 3, l_c3, p_numrows, 1);
  dbms_sql.define_array(c_select, 4, l_c4, p_numrows, 1);
  dbms_sql.define_array(c_select, 5, l_c5, p_numrows, 1);
  dbms_sql.define_array(c_select, 6, l_c6, p_numrows, 1);
  l_dummy := dbms_sql.execute(c_select);
  LOOP
    l_numrows := dbms_sql.fetch_rows(c_select);
    EXIT WHEN l_numrows < 1;
    dbms_sql.column_value(c_select, 1, l_c1);
    dbms_sql.column_value(c_select, 2, l_c2);
    dbms_sql.column_value(c_select, 3, l_c3);
    dbms_sql.column_value(c_select, 4, l_c4);
    dbms_sql.column_value(c_select, 5, l_c5);
    dbms_sql.column_value(c_select, 6, l_c6);
    -- here data conversion
    EXIT WHEN l_numrows < p_numrows;
  END LOOP;
  dbms_sql.close_cursor(c_select);
  c_insert := dbms_sql.open_cursor;
  dbms_sql.parse(c_insert, l_insert, dbms_sql.native);
  dbms_sql.bind_array(c_insert, ':c1', l_c1);
  dbms_sql.bind_array(c_insert, ':c2', l_c2);
  dbms_sql.bind_array(c_insert, ':c3', l_c3);
  dbms_sql.bind_array(c_insert, ':c4', l_c4);
  dbms_sql.bind_array(c_insert, ':c5', l_c5);
  dbms_sql.bind_array(c_insert, ':c6', l_c6);
  l_dummy := dbms_sql.execute(c_insert);
  dbms_sql.close_cursor(c_insert);
END;
```

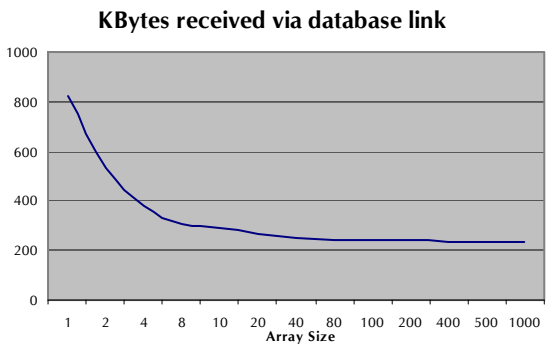
**Result of the Tests**
The values reported in the following table doesn't need a lot of remarks. The result is impressive. The network load is reduced 7 times. The execution time 27 times.

| Value | test1 | test2(2000) |
|---|---|---|
| Elapsed time in seconds | 122 | 4.5 |
| Net8 roundtrips to/from database link | 4009 | 10 |
| Kbytes sent via database link | 884 | 1 |
| Kbytes received via database link | 800 | 236 |
| Maximum UGA memory in Kbytes | 119 | 705 |

The Net8 overhead is higher then I had ever thought. According to me the strangest value is the number of bytes sent for the first test. In fact it's greater than the number of bytes received even if 200 Kbytes of data must be returned from the remote database.

**How Choice the Array Size?**
To help make this decision here are 2 graphics that shows the amount of bytes sent over the network (the real bottleneck) for an array size between 1 and 1000. The others statistics are not reported because their are similar.




As shown even with small values there is a performance gain. As well with an array size of 1 the performance are better than with the first test!?!?
Therefore a value of 200-500 will give good performance without using too many memory.

## Warning

At this time some bugs were found when the array interface is used over database link. For example inserting rows into a remote table cause a core dump.
Presently these bugs are not visible on Metalink because they are not public yes.

## New Feature Oracle8i

In Oracle8i there is a new feature named *bulk binds*. The *bulk binds* offer a simpler way to implement the array interface processing. In this article was not mentioned because this feature doesn't work over database links. Oracle Support reported that this problem would be fixed not before version 8.2.

## Conclusion

In PL/SQL as in SQL the simpler way is in many cases not the better from the performance point of view.