Ultra-High Performance SQL and PL/SQL in Batch Processing

Dr. Paul Dorsey
Dulcian, Inc.
www.dulcian.com

December 13, 2005

Overview

- The Problem:
  - Processing large amounts of data using SQL and PL/SQL poses unique challenges.
- The Story:
  - Traditional programming techniques cannot be effectively applied to large batch routines.
- The Real Life:
  - Organizations sometimes give up entirely in their attempts to use PL/SQL to perform large bulk operations!

ETL Tools

- "Bulk" idea (used by market leading ETL tools – Ab Initio or Informatica) :
  - copy large portions of a database to another location;
  - manipulate the data;
  - move it back.
- ETL vendors:
  - specialists at performing complex transformations → it works!
  - sub-optimal algorithm → it is expensive!
- Home-grown tools:
  - How to outperform the available ETL tools???
  - Different programming style of batch development!!!

DULCIAN INC.

Case Studies

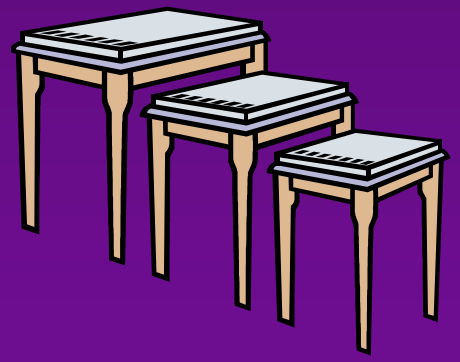
- 3 case studies with different scenarios:
  - 1. Multi-step complex transformation from source to target
  - 2. Periodic modification of a few columns in a database table with many columns
  - 3. Loading new objects into the database
- Presentation will discuss best practices in batch programming.

#1 Multi-Step Complex Transformation from Source to Target

- Classic data migration problem.
- 14 million objects → a complex set of transformations from source to target.
- Traditional coding techniques (Java and PL/SQL) → bad performance:
  - Java team
    - Pure OO-solution (Get/Set methods etc.)
    - One object per minute (~26.5 years to execute the month-end routine).
  - Same code refactored in PL/SQL
    - Exactly the same algorithm as the Java code
    - Significantly faster, but still would have required many days to execute.

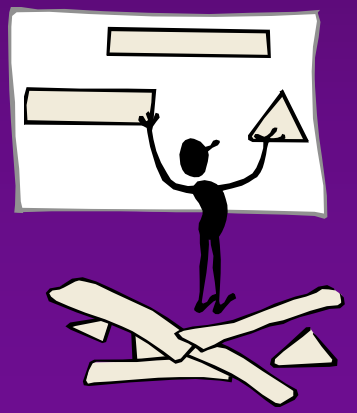
Case Study Test #1

- Table with only a few columns (3 character and 3 numeric)
- Load into a similar table while performing some transformations on the data. The new table will have a million records and be partitioned by the table ID (one of the numeric columns).
- Three transformations of the data will be shown to simulate the actual complex routine.



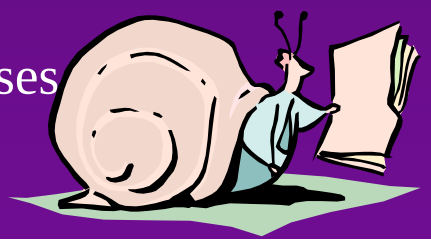
Sample Transformation

```
select a1,
     a1*a2,
     bc||de,
     de||cd,
     e-ef,
     to_date('20050301','YYYYMMDD'),--option A
     sysdate, -- Option B
     a1+a1/15,-- option A and B
     tan(a1), -- option C
     abs(ef)
from testB1
```



A. Complexity of Transformation Costs

- Varying parameters created very significant differences.
- Simple operations (add, divide, concatenate, etc.) had no effect on performance.
- Performance killers:
  - Function Calls (even built-it like **sysdate**):
    - Calls to **sysdate** in a SQL statement - no impact on performance.
    - Included in a loop can destroy performance
  - Complex calculations
    - This cost is independent of how records are processed.
    - Floating point operations are just slow (Calls to **tan()** or **ln()** take longer than inserting a record into the database)
    - 10g: **binary_float** data type that could help in some cases



DULCIAN INC.

B. Methods of Transformation Costs

- Various ways of moving the data were attempted.
  - Worst method = loop through a cursor FOR loop and use INSERT statements.
  - Even the simplest case takes about twice as long as other methods so some type of bulk operation was required.
  - Rule of thumb: 10,000 records/second using a cursor FOR loop method.

1. CREATE-TABLE-AS-SELECT (CTAS)

- Fairly fast mechanism
- For each step in the algorithm, create a global temporary table.
- Three sequential transformations still beat the cursor FOR loop by 50%.
- Note: adding a call to a floating point operation drastically impacted performance.
  - It took three times as long to calculate a TAN( ) and LN( ) as it did to move the data.

2. Bulk Load into Object Collections

- Load the data into memory (nested tables or VARRAY) and manipulate the data there.
- Problem: Exceeding the memory capacity of the server.
  - Massive collects are not well behaved.
  - Actually will run out of memory and crash. (ORA-600)
- Limit number of records to 250,000 at a time
  - Allows the routine to complete
  - Not very good performance.
  - Data must be partitioned for quick access.

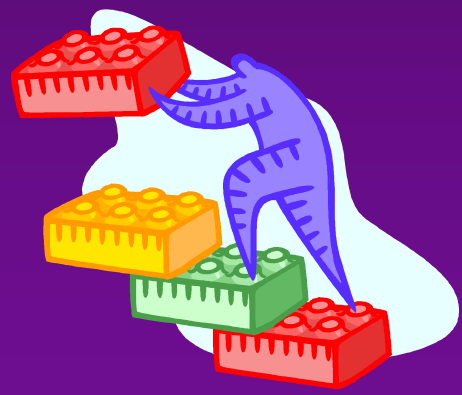
- Assuming no impact from partitioning, this method was still 60% slower than using CTAS.

3. Load Data into Object Collections N Records at a Time

- 1. Fetch 1000 records at once.
  - Simple loop used for transformation from one object collection to another. The last step was the second transformation from the object collection cast as a table.
  - Performed at same speed as CTAS.
- 2. Use FORALL
  - Oracle 9*i*, Release 2 - cannot work against object collections based on complex object types.
- Approach provided the best performance yet.
  - 8 seconds saved while processing 1 million records
  - Reduced overall processing speed to 42 seconds

4. Load Data into Object Collection 1 Record at a Time

- Use cursor FOR loop to load a COLLECT, then operated on the collection.
- Memory capacity exceeded unless number of records processed was limited.
- Even with limits, method did not perform significantly faster than using a simple cursor FOR loop.



Summary of results

Method	Extra	Data	A Simple	B + sysdate	C +sysdate +tan( )	D +sysdate +tan( )+ln( )
CTAS	2 buffer temp tables	1M	51	51	137	202
Full bulk load	Cast result into table	1M	Out of memory			
		4x250K	76	76	168	240
	Process second step as FOR-loop	1M	Out of memory			
		4x250K	56	56	148	220
Load data N records at a time; first step is BULK COLLECT LIMIT N	1000 rows per inserts	1M	54	54	126	219
	1000 rows per bulk, second step splits into the set of collections, Third step is FORALL	1M	42	42	135	206
Load data 1 record at a time; first step is regular loop	Next transformation via loop (full spin)	1M	Out of memory			
		4x250K	80	80	173	244
	Next transformation cast	1M	Out of memory			
		4x250K	96	96	188	260

Case Study Test #2

- Real case
  - Data - table with over 100 columns and 60 million records
  - Action - Each month, a small number of columns within these records needed to be updated.
  - Existing solution - update all 100 columns.
- Goal
  - find impact of sub-optimal code.
- Testing case
  - Source A = 126 columns, 5 columns with changed data.
  - Source B = 6 columns (5 columns with changed data and PK)
  - Target table being updated either had 5 or 126 columns.
  - Tried processing 1 and 2 million records.
  - Used the following syntax:

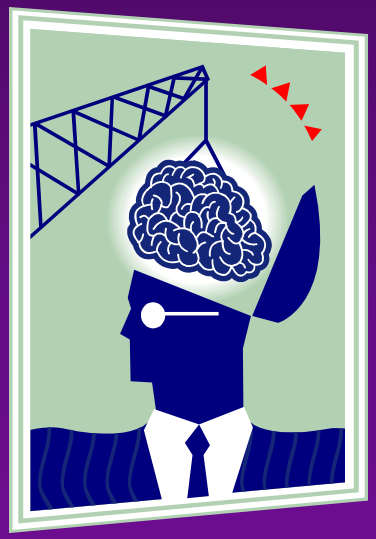
```
Update target t set (a,b,c,d,e)=
    (select a,b,c,d,e from source where oid = t.oid)
```

Results

- Updating 5 columns:
  - SQL is 50% faster on 6-column table (comparing to 126-column table)
  - PL/SQL is the slowest option.
- Updating all columns (unnecessarily):
  - on the 126-column table more than doubled processing time.

Lessons Learned

- Separate volatile and non-volatile data
- Only update the necessary columns.



Summary of Results

Method	Data	5 column target	126 column target

Update only 5 columns:
`Update target t`
`Set (a,b,c,d,e) =(select a,b,c,d,e`
`  from source where oid = t.oid)`
	2x1M	280	410

	2M	310	445

Update all columns:
`Update target t`
`Set (a,b,c,d,e) =(select a,b,c,d,e`
`  from source where oid = t.oid)`
	2x1M	N/A	970

Cursor spin:
`Declare`
`  Cursor c1 is`
`  Select *`
`  From source;`
`Begin`
`  For c in c1 loop`
`  Update target`
`  set a=c.a, …`
`  where oid = c.oid;`
`  End loop;`
`end;`
	2x1M	400	470

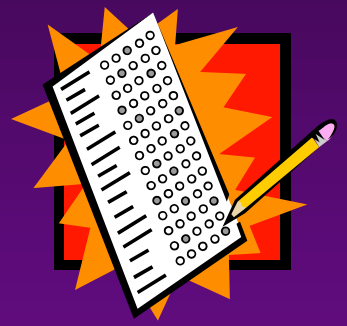
	2M	420	630

Case Study Test #3

- Several million new objects needed to be read into the system on a periodic basis.
- Objects enter system 120-column table
- Read from one source table → load into a number of tables at the same time (several parent/child pair):
  - Functionality not possible with most ETL tools
    - Most ETL tools write to one table at a time.
    - Need to write to parent table - then reread parent table for each child table to know where to attach child records

Test Structure

- Source table:
  - 120 columns
    - 40 number
    - 40 varchar2(1)
    - 40 varchar2 (2000) with populated default values
    - OID column – primary key
- Target tables:
  - Table A
    - ID
    - 40 varchar2(2000) columns
  - Table B
    - ID
    - 40 Number columns
    - Child of table A
  - Table C
    - 2 number columns, 2 varchar2 columns, 1 date column
    - child of table A



Test Methods

- Traditional method of spinning through a cursor
  - Poor performance
  - Generated an ORA-600 error.
  - Results worse than any other method tried.
- Bulk collecting limited number of records - best approach.
  - Best performance achieved with large limit (5000).
  - Conventional wisdom usually indicates that smaller limits are optimal.
- Simply using bulk operations does not guarantee success.
  - 1. Bulk collect the source data into an object collection, N rows at a time.
  - 2. Primary key of table A was generated.
  - 3. Three inserts of N rows were performed by casting the collection.
  - No better performance than the simple cursor FOR loop.
- Using bulk ForAll…Inserts
  - Performance much better - Half the time of the cursor FOR loop.
- Using “key table” to make lookups with cursor FOR loop faster.
  - No performance benefit to that approach.

Test Result Summary (1)

Method	Extra	Data	Timing
Loop source table → 3 consecutive inserts (commit each 10,000 records)		1M	ORA-600
		4x250K	508 sec
Bulk collect source data into object collection N rows at a time and generate A_OID (primary key of table A) → 3 inserts of N rows (cast the collection)	50 rows	1M	578 sec
		4x250K	564 sec
	100 rows	1M	558 sec
		4x250K	548 sec
	1000 rows	1M	522 sec
		4x250K	520 sec
	5000 rows	1M	503 sec
		4x250K	496 sec
	10000 rows	1M	512 sec
		4x250K	504 sec

Table Result Summary (2)

Method	Extra	Data	Timing
Bulk collect source data into set of object collections (one per each column) N rows at a time + generate A_OID (primary key of table A) → 3 inserts of N rows (FORALL … INSERT)	50 rows	1M	344 sec
		250K	336 sec
	100 rows	1M	317 sec
		250K	316 sec
	1000 rows	1M	271 sec
		250K	264 sec
	5000 rows	1M	263 sec
		250K	260 sec
	10000 rows	1M	265 sec
		250K	272 sec
Full insert with recording pairs (Source_ID; A_OID) into PL/SQL table. Next steps are querying that table to identify parent ID		1M	605 sec
		250K	480 sec

Conclusions

- Using “smart” PL/SQL can almost double performance speed.
- Keys to fast manipulation:
  - 1. Correct usage of bulk collect with a high limit (about 5000)
  - 2. ForAll…Insert
  - 3. Do not update columns unnecessarily.

Scripts used to create the tests
are available on the Dulcian website
(www.dulcian.com).

The J2EE SIG

Co-Sponsored by:

Chairperson – Dr. Paul Dorsey

About the J2EE SIG

- Mission: To identify and promote best practices in J2EE systems design, development and deployment.
- Look for J2EE SIG presentations and events at national and regional conferences
- Website: www.odtug.com/2005_J2EE.htm
- Join by signing up for the Java-L mailing list:
  - http://www.odtug.com/subscrib.htm

J2EE SIG Member Benefits

- Learn about latest Java technology and hot topics via SIG whitepapers and conference sessions.
- Take advantage of opportunities to co-author Java papers and be published.
- Network with other Java developers.
- Get help with specific technical problems from other SIG members and from Oracle.
- Provide feedback to Oracle on current product enhancements and future product strategies.

Share your Knowledge: Call for Articles/Presentations

- Submit articles, questions, … to

IOUG – The SELECT Journal

select@ioug.org

ODTUG – Technical Journal

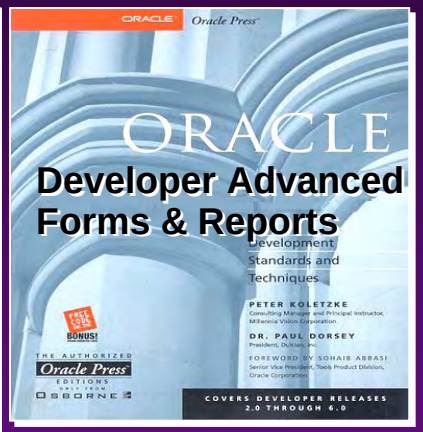
pubs@odtug.com

Dulcian's BRIM® Environment

- Full business rules-based development environment
- For Demo
  - Write "BRIM" on business card
- Includes:
  - Working Use Case system
  - "Application" and "Validation Rules" Engines

Contact Information

- Dr. Paul Dorsey – paul_dorsey@dulcian.com
- Michael Rosenblum – mrosenblum@dulcian.com
- Dulcian website - www.dulcian.com



Coming in 2006:
Oracle PL/SQL for Dummies