FAST MANIPULATION of DATA
and
E-MAIL DISTRIBUTION

Laurence Jennings

Board of Education Retirement System of the City of New York
65 Court Street, Brooklyn, NY 11201-4965

- Today's world of the internet, available everywhere and to everyone, has created instant arrival of news, videos, forums, chat rooms, and twitter through information pathways that never before existed. Business IT users expect these current norms to exist in the nine to five offices.

- Through information technology progressions, users have become naturally savvy and look toward IT departments to provide instantaneous client information through these same diverse forms of internet and intranet-based distributions. Instant adhocs, e-mails, and corporate dashboards, are now the new normal.

- In the constant quest to provide users real-time client information of balances, audit trails, delta reports, historical charts, fiscal comparison pie charts, histograms, daily details, inter systems feeds and critical reconciliations,

- We find Oracle the tool of choice which provides the backbone of extremely fast data manipulations to assist in our responses to users and their requests.

- Back office developers using Oracle SQL, PL/SQL and JAVA based coding are front line resources needed to be best apprised of the fastest data manipulation capabilities.

This presentation focuses on applying minimal code enhancement techniques to gain maximum data migration rates and collapse wall clock processing time-frames.

An overall review of Oracle resources in the behind the scenes operations needed for support are presented. We will cover basic aspects of the SGA(System Global Area) and key areas of buffer cache, library cache, various pools, and how they inter relate with SQL and PL/SQL.

We will also review code level nuances used to instigate parallel execution, bul inserts, updates, table index creates and updates used to enhance faster throughput.

This presentation focuses on applying minimal code enhancement techniques to gain maximum data migration rates and collapse wall clock processing time-frames.

Overview

- Oracle RAM resident Components (SGA)
- DBWR – Database Writer(s)
- LGWR – Log Writer
- DDL – Data Definition Language
- DML – Data Manipulation Language
- Stored Procedure
- Functions
- Parallelism (PL/SQL,SQL,Load Utilites)
- E-Mail – Distribution

SGA

- Oracle RAM resident Components
- SGA – System Global Area
- Buffer Pools
- Keep - DB_KEEP_CACHE
- Shared – DB_BUFFER_CACHE
- Default

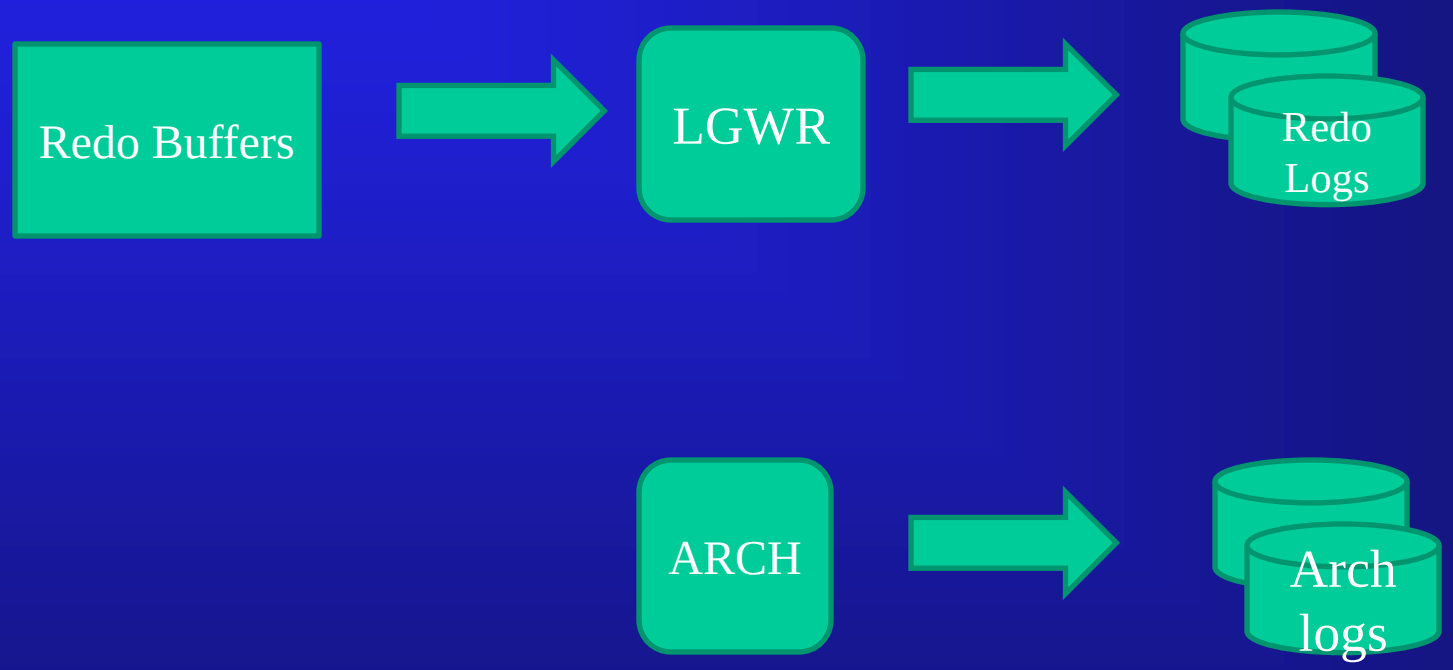
DBWR

- Dbwriter (DBWRn) process
- where n is multiple database writer
- Buffer Cache
- - Writes buffers to allows
- - buffer replacement
- - advance thread checkpoint
- Do forever
- { Scan LRU list
- Scan checkpoint queue
- Issues writes to disk datafile
- }

LGWR

- LGwriter (LGWR) process
- Flushes redo log buffer from SGA
- Redo Log Buffers
- - LGWR wakes up every 3 seconds
- - or when redo log buffer is full
- Do forever
- { Scan redo log buffers
- check capacity
- writes to on-line redo log files
- }

LGWR



External operations

1 - Tune SQL Insert DML

2 - Tune SQL*Loader

3 - Tune imports

4 - Use PL/SQL bulking

5 - Use solid-state media

Specific External operations

1 - **Tune SQL Insert DML** -

Parallelized insert programs,

each doing concurrent INSERT statements

(with enough freelists),

can speed up insert performance.

Specific External operations

2 - Tune SQL*Loader

-Using sqlldr Direct Load,

-and adjusting parameters

-improves INSERT performance.

Specific External operations

3 - **Tune imports**

- Use Oracle Data Pump (Formally Oracle import utility)
- Here are tips for hypercharging Oracle import.

Specific External operations

4 - Use PL/SQL bulking

PL/SQL often out-performs standard SQL inserts because of the array processing and bulking in the "forall" statement. Kent Crotty shows examples where *forall* provides a 30x performance improvement on inserts, making PL/SQL as fast as SQL*Loader, one of the fastest ways to load Oracle data.

Specific External Operations

5 - Use solid-state media

-Oracle with SSD can support

-over 500,000 rows per second

-for inserts,

-making it a great solution

-for datawarehouse

-ETL insert feeds.

DDL

- DDL – Data Definition Language

CREATE - to create objects in the database

ALTER - alters the structure of the database

DROP - delete objects from the database

TRUNCATE - remove all records from a table, including all spaces allocated for the records are removed

DML

- DML – Data Manipulation Language

SELECT

INSERT

UPDATE

DELETE

UNION

EXECUTE IMMEDIATE

LOCK TABLE - control concurrency

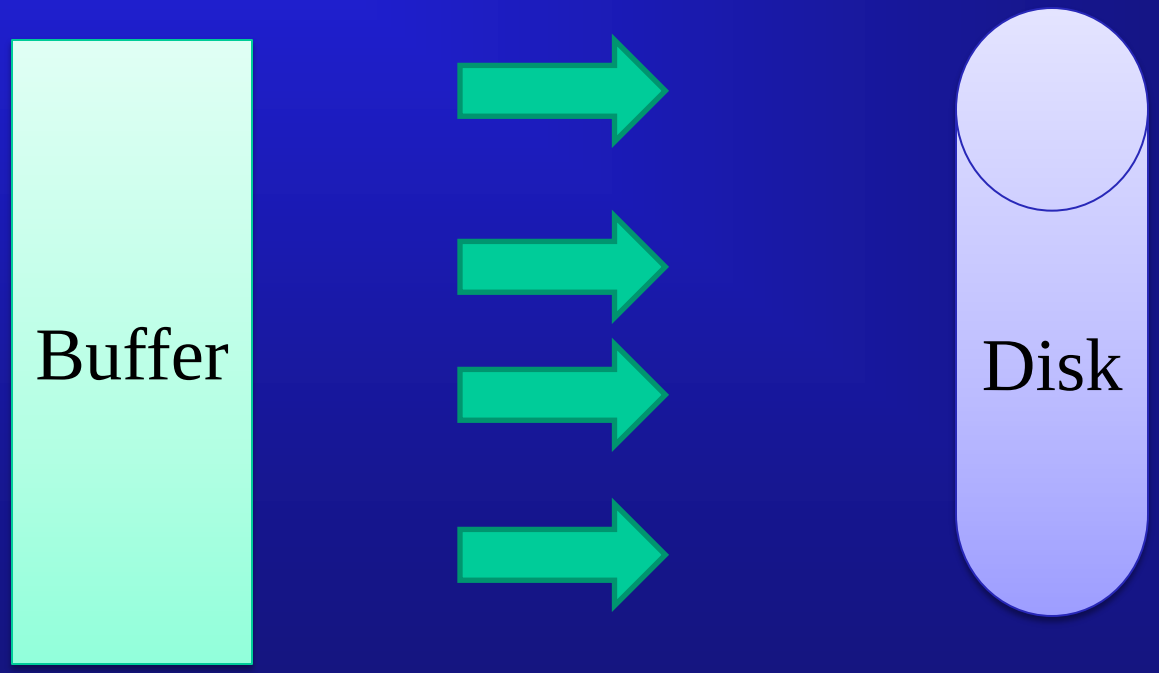
PARALLELISM

Using parallel execution,

Terabytes are processed in minutes

- Uses multiple processes for a single task
- Multiple (cores, I/O channels, bus architectures)

PARALLELISM



PARALLELISM

Degree of Parallelism

The parallel execution coordinator may enlist two or more of the instance's parallel execution servers to process a SQL statement. The number of parallel execution servers associated with a single operation is known as the **degree of parallelism**.

SELECT /*+ PARALLEL(employees 4) PARALLEL(departments 4) USE_HASH(employees) ORDERED */ MAX(salary), AVG(salary) FROM employees, depa

PARALLELISM

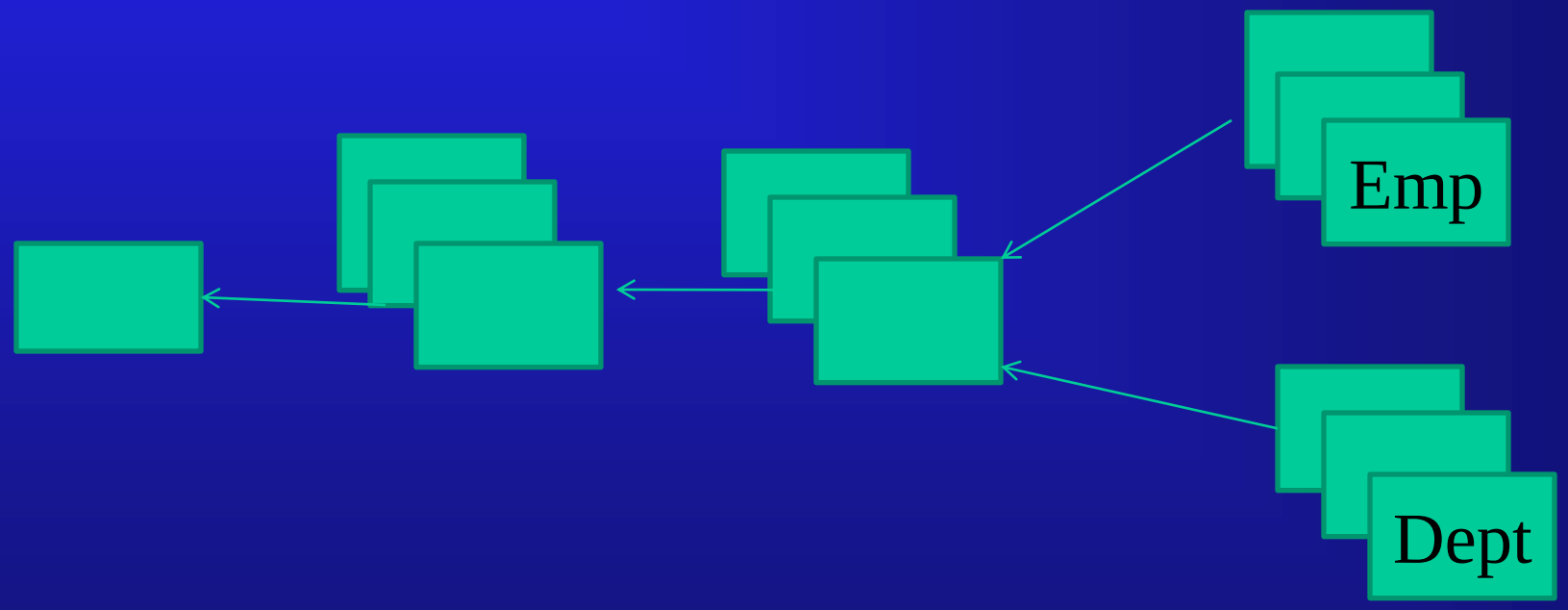
Dividing Work Among Parallel Execution Servers

```
 SELECT /*+ PARALLEL(employees 4)
PARALLEL(departments 4) USE_HASH(employees)
ORDERED */ MAX(salary), AVG(salary)
 FROM employees, departments
WHERE
  employees .department_id = departments .department_id
GROUP BY  employees .department_id;
```

```
SELECT /*+ PARALLEL(employees 4) PARALLEL(departments 4) USE_HASH(employees) ORDERED */ MAX(salary), AVG(salary) FROM employees, depa
```

PARALLELISM

Dividing Work Among Parallel Execution Servers



PEC **Group by** **Hash Join** **Full Scans**

PARALLELISM

- Restrictions:
- Parallel DML operation cannot be done on tables with triggers
- Certain constraints
- Self-referential integrity
- A type of foreign key references a parent key in the same table.
- Delete cascade
- Distributed transactions as database links

Self Referential Integrity

Primary Key
of referenced table

Foreign Key
(values in dependent table must match a value in unique key or primary key of referenced table)

Dependent or Child Table

Referenced or Parent Table

Table EMP

EMPNO	ENAME	JOB	MGR	HIREDATE	SAL	COMM	DEPTNO
7329	SMITH	CEO	7329		9,000.00		20
7499	ALLEN	VP–SALES	7329		7,500.00	100.00	30
7521	WARD	MANAGER	7499		5,000.00	200.00	30
7566	JONES	SALESMAN	7521		2,975.00	400.00	30

INSERT INTO

7571	FORD	MANAGER	7331	23–FEB–90	5,000.00	200.00	30

This row violates the referential constraint, because "7331" is not present in the referenced table's primary key; therefore, it is not allowed in the table.

PARALLELISM

- Restrictions:
- DML on tables with self-referential integrity constraints is not parallelized if the referenced keys (primary keys) are involved.
- For DML on all other columns, parallelism is possible.

PARALLELISM

- Restrictions:
- Delete on tables having a foreign key with delete cascade is not parallelized because parallel execution servers will try to delete rows from multiple partitions (parent and child tables).

PARALLELISM

- **Trigger Restrictions**
- A DML operation will not be parallelized if the affected tables contain enabled triggers that may get fired as a result of the statement.
- This implies that DML statements on tables that are being replicated will not be parallelized.

PARALLELISM

- In this example, the DML statement queries a remote object:

- INSERT /*+ APPEND PARALLEL (t3,2) */ INTO t3 SELECT * FROM t4@dblink;

- The query operation is executed serially without notification because it references a remote object.

PARALLELISM

- In this example, the DML statement queries a remote object:
- DELETE /*+ PARALLEL (t1, 2) */ FROM t1@dblink;
- The DELETE operation is not parallelized because it references a remote object

PARALLELISM

- DML statement queries a remote object:
- SELECT * FROM t1@dblink; DELETE /*+ PARALLEL (t2,2) */ FROM t2; COMMIT;
- The DELETE operation is not parallelized because it occurs in a distributed transaction (which is started by the SELECT statement)

PARALLELISM
using
Oracle Utilities

PARALLELISM

- Import Operations:
- IMP (Import)
- IMPDP (Data pump)
- SQLLDR (SQL*Loader)

PARALLELISM

- IMP **IMPORT:**
- Create an indexfile so that you can create indexes AFTER you have imported data.
- Do this by setting INDEXFILE to a filename and then import.
- No data will be imported but a file containing index definitions will be created.

PARALLELISM

- IMP continued…
- Place the file to be imported on a separate physical disk from the oracle data files
- Increase DB_CACHE_SIZE (DB_BLOCK_BUFFERS prior to 9i)
- Set the LOG_BUFFER to a big value and restart oracle.

PARALLELISM

- IMP continued…
- Stop redo log archiving if it is running (ALTER DATABASE NOARCHIVELOG;)
- Create a BIG tablespace with a BIG rollback segment inside. Set all other rollback segments offline .

PARALLELISM

- IMP continued…
- Use COMMIT=N in the import parameter file if you can afford it
- Use STATISTICS=NONE in the import parameter file to avoid time consuming to import the statistics
- Remember to run the indexfile previously created

PARALLELISM

- IMP continued…
- Use COMMIT=N in the import parameter file if you can afford it
- Use STATISTICS=NONE in the import parameter file to avoid time consuming to import the statistics
- Remember to run the indexfile previously created

PARALLELISM

- IMPDP
- impdp hr/hr DIRECTORY=dpump_dir1 DUMPFILE=expfull.dmp FULL=y PARALLEL=35
- TABLE_EXISTS_ACTION={SKIP | APPEND | TRUNCATE | REPLACE}

HYPERCHARGING PL/SQL

HYPERCHARGING PL/SQL

- PL/SQL forall operator speeds 30x faster for table inserts.
- Kent Crotty, author of Easy Oracle Application Express (HTML-DB) conducted a study to prove the speed of the PL/SQL forall over vanilla SQL inserts, and found that FORALL was 30x faster in his small test:

HYPERCHARGING PL/SQL

- DECLARE
- TYPE prod_tab IS
- TABLE OF products%ROWTYPE;
- products_tab prod_tab := prod_tab();
- start_time number;
- end_time number;

HYPERCHARGING PL/SQL

- BEGIN
- SELECT * BULK COLLECT INTO products_tab
- FROM products;
- EXECUTE IMMEDIATE
- 'TRUNCATE TABLE products';

- Start_time := DBMS_UTILITY.get_time;

HYPERCHARGING PL/SQL

- Start_time := DBMS_UTILITY.get_time;
- FOR i in products_tab.first .. products_tab.last LOOP INSERT INTO products (product_id, product_name) VALUES (products_tab(i).product_id, products_tab(i).product_name);
- END LOOP;
- end_time := DBMS_UTILITY.get_time;
- DBMS_OUTPUT.PUT_LINE
- (‘Conventional Insert: ’||to_char(end_time-start_time));

HYPERCHARGING PL/SQL

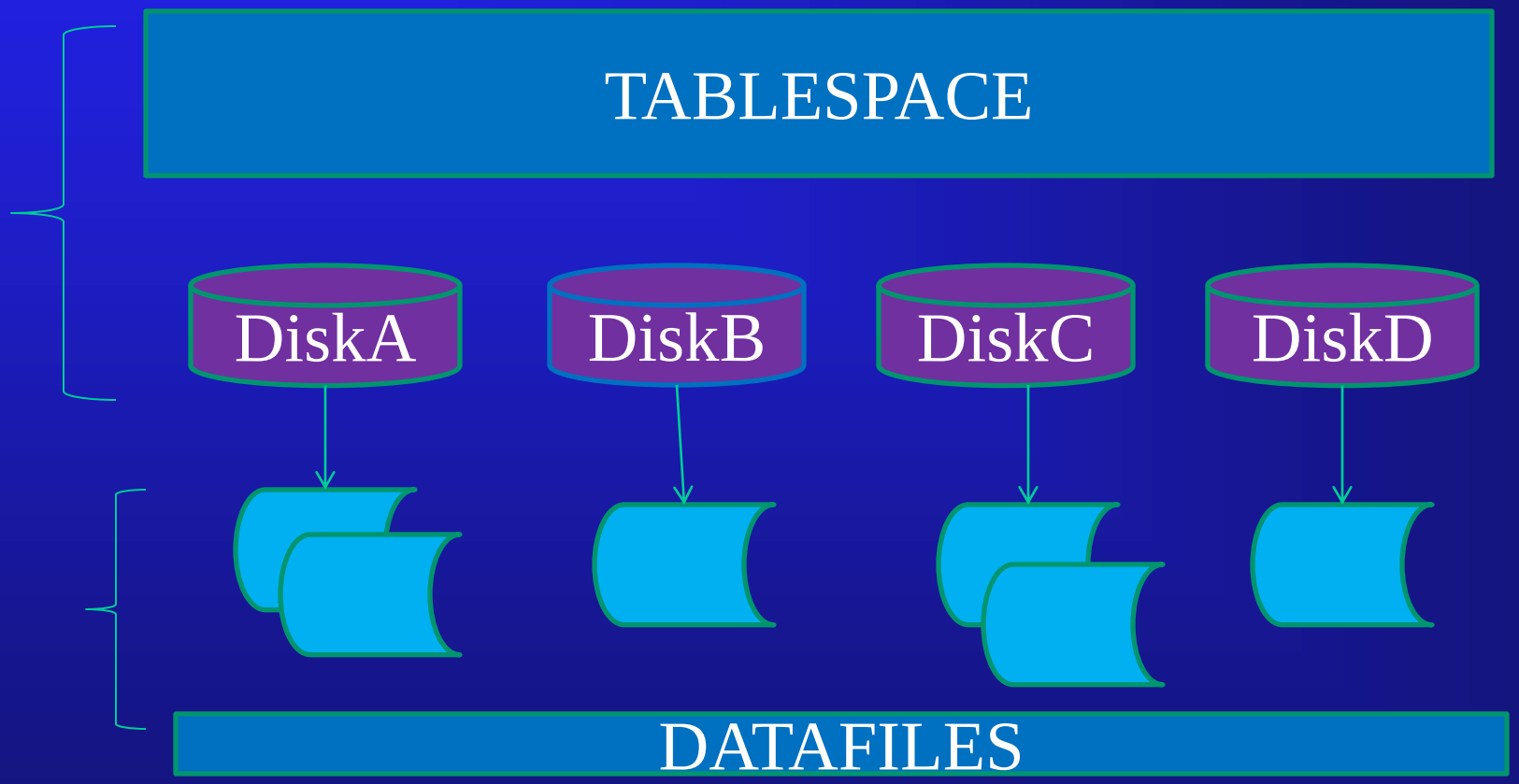
- EXECUTE IMMEDIATE 'TRUNCATE TABLE products';
- Start_time := DBMS_UTILITY.get_time;
- FORALL i in products_tab.first .. products_tab.last
- INSERT INTO products VALUES products_tab(i);
- end_time := DBMS_UTILITY.get_time;
- DBMS_OUTPUT.PUT_LINE(‘Bulk Insert: ’||to_char(end_time-start_time));
- COMMIT;
- END;

HYPERCHARGING PL/SQL

- Conventional Insert: 686
- Bulk Insert: 22

- **Oracle**
- **SHARING EVERYTHING**
- **NO DATA PARTITIONING REQUIRED**

Share Everything
No data partitioning required



PARALLELISM

- SQL*Loader
- To speed up your loads, you can use a parallel direct-path load

- SQLLDR USERID=scott/tiger control=load1 DIRECT=TRUE PARALLEL=TRUE

DEVELOPING APPROACH

DEVELOPING APPROACH

- Pseudocode the required functionality
- Category, Quantify, test cases
- Build test code to play test cases
- Use FUNCTIONS, and PACKAGES
- Repeat the steps…

DML

- Create or replace procedure NYOUG_e_mail_distribution AS
- Mail_to varchar2(30);
- Mail_from varchar2(30);
- Subject varchar2(100);
- Msg varchar2(4000);
- User1 varchar2(30) :=‘user1@xxx.gov’;
- User2 varchar2(30):=‘user2@xxx.gov’;

- BEGIN

- EXECUTE IMMEDIATE
- ‘ ALTER Temp_tablex parallel 15 nologging’;

- EXECUTE IMMEDIATE
- ‘TRUNCATE TABLE Temp_tablex’;

- BEGIN
- EXECUTE IMMEDIATE ‘ ALTER Temp_tablex parallel 15 nologging’;
- EXECUTE IMMEDIATE ‘TRUNCATE TABLE Temp_tablex’;
- INSERT ALL /*+ APPEND */ or /*+ APPEND_VALUES */
- INTO Temp_tablex
- VALUES (member,ssn,fname,lname,pension_num,’R’)
- SELECT * FROM Pension_tablex;
- EMAIL_HANDLER.send_email(mail_from, mail_to,subject,msg);

FAST E-MAIL DISTRIBUTIONS

FAST E-MAIL DISTRIBUTIONS

- http://www.orafaq.com/wiki/Send_mail_from_PL/SQL

- **Contents**
- 1 Send mail with UTL_MAIL
- 2 Send mail with UTL_SMTP
- 3 Send mail with UTL_SMTP - with attachments
- 4 Send mail with UTL_TCP
- 5 Send mail with UTL_TCP - with attachments

FAST E-MAIL DISTRIBUTIONS

```
• Create or replace procedure NYOUG_e_mail_distribution   AS
• Mail_to        varchar2(30);
• Mail_from      varchar2(30);
• Subject        varchar2(100);
• Msg            varchar2(4000);
• User1          varchar2(30) :='user1@xxx.gov';
• User2          varchar2(30):='user2@xxx.gov';
• BEGIN
• EXECUTE IMMEDIATE ' ALTER Temp_tablex parallel 15 nologging';
• EXECUTE IMMEDIATE 'TRUNCATE TABLE Temp_tablex';
• INSERT ALL /*+ APPEND */   or /*+ APPEND_VALUES */
• INTO    Temp_tablex
• VALUES (member,ssn,fname,lname,pension_num,'R')
• SELECT * FROM  Pension_tablex;
• EMAIL_HANDLER.send_email(mail_from, mail_to,subject,msg);
```

- BEGIN
- EXECUTE IMMEDIATE ‘ ALTER Temp_tablex PARALLEL 15 NOLOGGING’;
- EXECUTE IMMEDIATE ‘TRUNCATE TABLE Temp_tablex’;

- INSERT ALL /*+ APPEND */ or /*+ APPEND_VALUES */
- INTO Temp_tablex
- VALUES (username,mailto_email,mailfrom_email)
- SELECT * FROM Pension_tablex;
- ……
- …….
- ……
- FOR cur_distrubution in
- (SELECT Username,mailto_email, mailfrom_email
- FROM Temp_tablex
- ORDER BY username)
- LOOP
- mail_to :=cur_dist.username;
- salutation :='AAAAA';
- EMAIL_HANDLER.send_email(mail_from, mail_to, subject, msg);
- END LOOP;
- END E_MAIL_DISTRIBUTION;
-

- BERS, at ZZTOP46@bers.nyc.gov

- ---------- Interface Audit report ----------
- Step1 - Interface Extract Started... TUESDAY 20-SEP-11 07:45:42
- Step2 - Interface 9,136,835 staging created TUESDAY 20-SEP-11 07:45:42
- Step3 - Interface 9,136,835 staging rolled to CPMS_CURRENT TUESDAY 20-SEP-11 07:45:48
- Step4 - Interface 9,136,835 Prior day rolled to CPMS_PRIOR TUESDAY 20-SEP-11 07:45:49
- Step5 - Interface 9,136,835 Extract Records created... TUESDAY 20-SEP-11 07:45:49
- Step6 - Differential table NULLS for no chgs and actuals TUESDAY 20-SEP-11 07:45:49
- Step7 - New records found to changed records in the DELTAS TUESDAY 20-SEP-11 07:45:49
- Step8 - Archive todays Interface Differentials for that day TUESDAY 20-SEP-11 07:45:49
- Step9 - Delta records RECAP of Enrollment changes TUESDAY 20-SEP-11 07:45:49
- Step10 - Enrollment Interface TUESDAY 20-SEP-11 07:45:50
- Step11 - Report Current, prior and New Deltas counts TUESDAY 20-SEP-11 07:45:50
- Step12 - e-mail reports to managers, users via routing list TUESDAY 20-SEP-11 07:45:50
- Step13 - Interface Extract Ended... TUESDAY 20-SEP-11 07:45:50

- -- THIS WEEKS Daily Incrementals Delivered to Onbase, and reported to Auditing ----------
- Mon ==> \\2mtBERFILEAPP\Shared\Auditing\Monday_incremental_Audit_output.txt
- Tue ==> \\2mtBERFILEAPP\Shared\Auditing\Tuesday_incremental_Audit_output.txt
- Wed ==> \\2mtBERFILEAPP\Shared\Auditing\Wednesday_incremental_Audit_output.txt
- Thur ==> \\2mtBERFILEAPP\Shared\Auditing\Thursday_incremental_Audit_output.txt
- Fri ==> \\2mtBERFILEAPP\Shared\Auditing\Friday_incremental_Audit_output.txt
-
- -- LAST WEEKS Daily Incrementals Delivered to Onbase, and reported to Auditing ----------
- Mon ==> \\2mtBERFILEAPP\Shared\Auditing\Lst_Monday_incremental_Audit_output.txt
- Tue ==> \\2mtBERFILEAPP\Shared\Auditing\Lst_Tuesday_incremental_Audit_output.txt
- Wed ==> \\2mtBERFILEAPP\Shared\Auditing\Lst_Wednesday_incremental_Audit_output.txt
- Thu ==> \\2mtBERFILEAPP\Shared\Auditing\Lst_Thursday_incremental_Audit_output.txt
- Fri ==> \\2mtBERFILEAPP\Shared\Auditing\Lst_Friday_incremental_Audit_output.txt
-
- --- V3 INTERFACE Output ---
- CPMS DELTAS ==> \\2MTBERFILEAPP\Shared\Auditing\CPMS_Incrementals\CPMS_Incremental.txt

E-Mail Distribution

- E-Mail - Distribution