

Exciting SQL/PLSQL new features

By

Riyaj shamsudeen

Who am I?

- 15 years using Oracle
 - Over 12 years as Oracle DBA
 - Certified DBA versions 7.0,7.3,8,8i &9i
 - Specializes in performance tuning and Internals
 - Currently working for JCPenney
 - Assortment planning & Allocation systems
 - Adjunct faculty – Northlake College
 - Email: [rshamsud at gmail.com](mailto:rshamsud@gmail.com)
[rshams4 at yahoo.com](mailto:rshams4@yahoo.com)
-

Sub query factoring

Sub query factoring

- Results of a subquery can be accessed as if it is another row source.
 - Even nesting of subqueries supported.
 - Improved performance as aggregation can be done once and reused.
-

Sub query factoring

□ WITH clause

Defines dept_avg row source

WITH

dept_avg as

(select deptno, avg(sal) avg_sal from emp group by deptno),

all_avg as

(select avg(avg_sal) avg_sal from **dept_avg**)

select d.deptno, d_avg.avg_sal , a_avg.avg_sal

from

dept d,

dept_avg d_avg,

all_avg a_avg

where

d_avg.avg_sal >= a_avg.avg_sal and

d.deptno=d_avg.deptno

/

Dept_avg defined above

Query accessing row sources

Explain plan..

Temporary table creation for dept_avg subquery

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	52
1	TEMP TABLE TRANSFORMATION			
2	LOAD AS SELECT			
3	SORT GROUP BY		14	364
4	TABLE ACCESS FULL	EMP	14	364
* 5	HASH JOIN		1	52
6	NESTED LOOPS		1	39
7	VIEW		1	13
8	SORT AGGREGATE		1	13
9	VIEW		14	182
10	TABLE ACCESS FULL	SYS_TEMP_0FD9D6604_740118	14	364
* 11	VIEW		1	26
12	TABLE ACCESS FULL	SYS_TEMP_0FD9D6604_740118	14	364
13	TABLE ACCESS FULL	DEPT	4	52

Temporary table accessed

Lines from trace file

- **Internally, Oracle is creating SQL specific global temporary table. This table disappeared immediately after SQL execution.**

```
CREATE GLOBAL TEMPORARY TABLE "SYS"."SYS_TEMP_0FD9D6605_740118"  
("C0" NUMBER(2),"C1" NUMBER ) IN_MEMORY_METADATA  
CURSOR_SPECIFIC_SEGMENT STORAGE (OBJNO 4254950917 )  
NOPARALLEL
```

Subquery factoring with dblink

- Works even with database links.
 - You can materialize the remote table locally and then join
 - If the tables are remote and accessed numerous times, avoids extra work.
-

Subquery factoring with dblink

With dept_avg as

```
(select deptno, avg(sal) avg_sal from emp@loopback group by  
deptno ),
```

all_avg as

```
(select avg(avg_sal) avg_sal from dept_avg )  
select d.deptno, d_avg.avg_sal , a_avg.avg_sal  
from  
dept d,  
dept_avg d_avg,  
all_avg a_avg
```

where

```
d_avg.avg_sal >= a_avg.avg_sal and  
d.deptno=d_avg.deptno
```

```
/
```

Subquery factoring with dblink

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	42
1	TEMP TABLE TRANSFORMATION			
2	LOAD AS SELECT			
3	SORT GROUP BY		3	78
4	REMOTE		14	364
* 5	HASH JOIN		1	42
6	NESTED LOOPS		1	39
7	VIEW		1	13
8	SORT AGGREGATE		1	13
9	VIEW		3	39
10	TABLE ACCESS FULL	SYS_TEMP_0FD9D6602_8103BDC3	3	78
* 11	VIEW		1	26
12	TABLE ACCESS FULL	SYS_TEMP_0FD9D6602_8103BDC3	3	78
13	TABLE ACCESS FULL	DEPT	4	12

Remote SQL Information (identified by operation id):

4 - SELECT "SAL","DEPTNO" FROM "EMP" "EMP" (accessing 'LOOPBACK')

Subquery factoring..

- ❑ **Oracle can decide not to materialize temporary table.**

with dept_avg as

```
(select deptno, avg(sal) avg_sal from emp group by deptno ),
```

all_avg as

```
(select avg(sal) avg_sal from emp )
```

```
select d.deptno, d_avg.avg_sal , a_avg.avg_sal
```

```
from
```

```
dept d,
```

```
dept_avg d_avg,
```

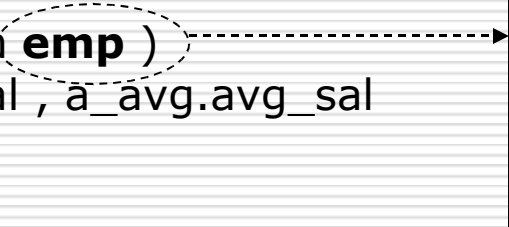
```
all_avg a_avg
```

where

```
d_avg.avg_sal >= a_avg.avg_sal and
```

```
d.deptno=d_avg.deptno
```

```
/
```



We access emp directly instead of accessing dept_avg sub query

Subquery factoring..

Missing "Temp table transformation" step:

EXPLAIN PLAN

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	52
* 1	HASH JOIN		1	52
2	NESTED LOOPS		1	39
3	VIEW		1	13
4	SORT AGGREGATE		1	13
5	TABLE ACCESS FULL	EMP	14	182
* 6	VIEW		1	26
7	SORT GROUP BY		14	364
8	TABLE ACCESS FULL	EMP	14	364
9	TABLE ACCESS FULL	DEPT	4	52

Subquery factoring..

- Undocumented (so unsupported) **Materialize hint** can be used to force creation of temporary table: (Thanks Jonathan lewis)

With dept_avg as

```
(select /*+ materialize */ deptno, avg(sal) avg_sal from emp group by deptno ),...
```

Id	Operation	Name	Rows	Bytes
0	SELECT STATEMENT		1	52
1	TEMP TABLE TRANSFORMATION			
2	LOAD AS SELECT			
3	SORT GROUP BY		14	364
4	TABLE ACCESS FULL	EMP	14	364
5	LOAD AS SELECT			
6	SORT AGGREGATE		1	13
7	TABLE ACCESS FULL	EMP	14	182
* 8	HASH JOIN		1	52
9	NESTED LOOPS		1	39
10	VIEW		1	13
11	TABLE ACCESS FULL	SYS_TEMP_0FD9D660D_740118	1	13
* 12	VIEW		1	26
13	TABLE ACCESS FULL	SYS_TEMP_0FD9D660C_740118	14	364
14	TABLE ACCESS FULL	DEPT	4	52

Subquery factoring..

- In read only databases, even with materialize hint, Oracle will not create temporary table.

```
Rows      Row Source Operation
-----
2  HASH JOIN  (cr=10 pr=0 pw=0 time=3483 us)
2  NESTED LOOPS (cr=6 pr=0 pw=0 time=1005 us)
1  VIEW      (cr=3 pr=0 pw=0 time=552 us)
1  SORT AGGREGATE (cr=3 pr=0 pw=0 time=543 us)
14  TABLE ACCESS FULL OBJ#(20979) (cr=3 pr=0 pw=0 time=365 us)
2  VIEW      (cr=3 pr=0 pw=0 time=451 us)
3  SORT GROUP BY (cr=3 pr=0 pw=0 time=445 us)
14  TABLE ACCESS FULL OBJ#(20979) (cr=3 pr=0 pw=0 time=141 us)
4  TABLE ACCESS FULL OBJ#(20980) (cr=4 pr=0 pw=0 time=117 us)
```

Connect by enhancements

Connect by enhancements

- Prior to 9i, connect by query always resulted in Nested loops join / Filter operations.
 - From 9i, optimizer considers other joins options also.
 - New connect by pump step in the explain plan.
-

Connect by..

Consider the following query:

```
select lpad(' ',2*level-1 ,ename) ename
from emp
connect by prior empno = mgr
start with mgr is null
/
```

Connect by -8i plan

Execution Plan

```
0      SELECT STATEMENT Optimizer=CHOOSE
1    0      CONNECT BY
2    1        TABLE ACCESS (FULL) OF 'EMP'
3    1        TABLE ACCESS (BY USER ROWID) OF 'EMP'
4    1        TABLE ACCESS (FULL) OF 'EMP'
```

Statistics

```
0 recursive calls
180 db block gets
89 consistent gets ←
0 physical reads
0 redo size
692 bytes sent via SQL*Net to client
358 bytes received via SQL*Net from client
2 SQL*Net roundtrips to/from client
0 sorts (memory)
0 sorts (disk)
14 rows processed
```

180 + 89 = 269 buffer gets

Connect by -10g plan

Execution Plan

```
0      SELECT STATEMENT Optimizer=ALL_ROWS (Cost=2 Card=14 Bytes=196)
1      0      CONNECT BY (WITH FILTERING)
2      1      FILTER
3      2      TABLE ACCESS (FULL) OF 'EMP' (TABLE) (Cost=2 Card=14 Bytes=196)
4      1      HASH JOIN
5      4      CONNECT BY PUMP
6      4      TABLE ACCESS (FULL) OF 'EMP' (TABLE) (Cost=2 Card=14 Bytes=196)
7      1      TABLE ACCESS (FULL) OF 'EMP' (TABLE) (Cost=2 Card=14 Bytes=196)
```

Statistics

```
1  recursive calls
0  db block gets
15 consistent gets
0  physical reads
0  redo size
716 bytes sent via SQL*Net to client
664 bytes received via SQL*Net from client
2  SQL*Net roundtrips to/from client
5  sorts (memory)
0  sorts (disk)
14 rows processed
```

**15 buffer gets in 10g,
Compared to 269 buffer gets in 8i**

Connect by

- Interestingly, this feature is available in 8i, but disabled by default.

`_new_connect_by_enabled=true`

: enables this behavior in 8i

- Of course, this feature can be disabled in 10g, by setting

`_old_connect_by_enabled=true`

Connect by -old

```
update /* old=true */ connect_by_perf
  set comment_filler=substr(comment_filler, 1, 50)
  where id in
    (select id from connect_by_perf
     start with id =407252
     connect by parent_id = prior id )
```

103899+26893=130792

Buffer gets

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.01	0.00	0	0	0	0
Execute	1	3.70	3.61	0	103899	25893	25453
Fetch	0	0.00	0.00	0	0	0	0
total	2	3.71	3.61	0	103899	25893	25453

Rows	Row	Source	Operation
0	0	UPDATE	(cr=103899 pr=0 pw=0 time=3610284 us)
25453	25453	HASH JOIN RIGHT SEMI	(cr=103899 pr=0 pw=0 time=2065951 us)
25453	25453	VIEW	(cr=101816 pr=0 pw=0 time=1527339 us)
25453	25453	CONNECT BY	(cr=101816 pr=0 pw=0 time=1501883 us)
1	1	INDEX UNIQUE SCAN	CP_PK (cr=3 pr=0 pw=0 time=101 us)(object id 37347)
1	1	TABLE ACCESS BY USER ROWID	CONNECT_BY_PERF (cr=1 pr=0 pw=0 time=110 us)
25452	25452	TABLE ACCESS BY INDEX ROWID	CONNECT_BY_PERF (cr=101812 pr=0 pw=0 time=1301315 us)
25452	25452	INDEX RANGE SCAN	CP_PARENT_ID (cr=76360 pr=0 pw=0 time=897992 us)(object id 37348)
839953	839953	TABLE ACCESS FULL	CONNECT_BY_PERF (cr=2083 pr=0 pw=0 time=393 us)

Connect by-new

```
update /* old=false */ connect_by_perf
  set comment_filler=substr(comment_filler, 1, 50)
  where id in
    (select id from connect_by_perf
     start with id =407252
     connect by parent_id = prior id )
```

53279+25453=78732 buffer gets

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	1.92	1.86	0	53279	25453	25453
Fetch	0	0.00	0.00	0	0	0	0
total	2	1.92	1.87	0	53279	25453	25453

Rows	Row Source Operation
0	UPDATE (cr=53279 pr=0 pw=0 time=1866313 us)
25453	HASH JOIN RIGHT SEMI (cr=53279 pr=0 pw=0 time=1299514 us)
25453	VIEW (cr=51196 pr=0 pw=0 time=860377 us)
25453	CONNECT BY WITH FILTERING (cr=51196 pr=0 pw=0 time=834917 us)
1	TABLE ACCESS BY INDEX ROWID OBJ#(37346) (cr=4 pr=0 pw=0 time=160 us)
1	INDEX UNIQUE SCAN OBJ#(37347) (cr=3 pr=0 pw=0 time=89 us)(object id 37347)
25452	NESTED LOOPS (cr=51192 pr=0 pw=0 time=645889 us)
25453	BUFFER SORT (cr=0 pr=0 pw=0 time=57928 us)
25453	CONNECT BY PUMP (cr=0 pr=0 pw=0 time=67 us)
25452	TABLE ACCESS BY INDEX ROWID OBJ#(37346) (cr=51192 pr=0 pw=0 time=557955 us)
25452	INDEX RANGE SCAN OBJ#(37348) (cr=50966 pr=0 pw=0 time=446449 us)(object id 37348)
0	TABLE ACCESS FULL OBJ#(37346) (cr=0 pr=0 pw=0 time=0 us)
839953	TABLE ACCESS FULL OBJ#(37346) (cr=2083 pr=0 pw=0 time=372 us)

Sys_connect_by_path

□ **A new function introduced:**

```
select
```

```
  lpad(' ',2*level-1)||sys_connect_by_path (ename, '/')
```

```
from emp
```

```
connect by prior empno = mgr ;
```

```
/king
```

```
  /king/jones
```

```
    /king/jones/scott
```

```
      /king/jones/scott/adams
```

```
    /king/jones/ford
```

```
      /king/jones/ford/smith
```

```
/king/blake
```

```
  /king/blake/allen
```

```
    /king/blake/ward
```

```
      /king/blake/martin
```

```
        /king/blake/turner
```

```
          /king/blake/james
```

```
/king/clark
```

```
  /king/clark/miller
```

Data without this function

king	jones	scott	adams
	ford	smith	
	blake	allen	
		ward	
		martin	
		turner	
		james	
	clark	miller	

Connect by enhancements

- Few new pseudo columns in 10g
 - connect_by_iscycle
 - connect_by_isleaf
-

Handling loops

- Loops can be effectively handled

```
select
```

```
  lpad(' ',2*level-1)||sys_connect_by_path (ename, '/') ename
from emp
connect by prior empno = mgr
start with empno=7782
```

```
/
```

```
  EMPNO  ENAME
```

```
-----
```

```
  7782   /CLARK
```

```
  7934   /CLARK/MILLER
```

```
REM updating to create a loop..
```

```
update emp set mgr=7934 where  ename='CLARK'
```

```
/
```

Handling loops -8i

Loop throws error:

```
select
```

```
  lpad(' ',2*level-1)||sys_connect_by_path (ename, '/') ename
```

```
from emp
```

```
  connect by prior empno = mgr
```

```
  start with empno=7782
```

```
SQL> /
```

```
ERROR:
```

```
ORA-01436: CONNECT BY loop in user data
```

Handling loops-10g

```
select
  lpad(' ',2*level-1)||sys_connect_by_path (ename, '/') ename,
  connect_by_iscycle
from emp
connect by nocycle prior empno = mgr
start with empno=7782
/
```

ENAME	CONNECT_BY_ISCYCLE
/CLARK	0
/CLARK/MILLER	1

Connect_by_isleaf

Connect_by_isleaf pseudo column

1 means leaf 0 – not a leaf row (or a parent)

```
select
lpad(' ',2*level-1)||sys_connect_by_path (ename, '/') ename, connect_by_isleaf
from emp
connect by prior empno = mgr
start with mgr is null
/
```

EName	CONNECT_BY_ISLEAF
/KING	0
/KING/JONES	0
/KING/JONES/SCOTT	0
/KING/JONES/SCOTT/ADAMS	1
/KING/JONES/FORD	0
/KING/JONES/FORD/SMITH	1
/KING/BLAKE	0
/KING/BLAKE/ALLEN	1
/KING/BLAKE/WARD	1
/KING/BLAKE/MARTIN	1
/KING/BLAKE/TURNER	1
/KING/BLAKE/JAMES	1

Model



Try this SQL?

```
select item, location, week, inventory, sales_qty, rcpt_qty , sales_so_for,
       rcpt_so_for
from item_data
model return updated rows
partition by (item)
dimension by (location, week)
measures ( 0 inventory , sales_qty, rcpt_qty, 0 sales_so_for, 0 rcpt_so_for )
rules (
inventory [location, week] = nvl(inventory [cv(location), cv(week)-1 ] ,0)
                        - sales_qty [cv(location), cv(week) ] +
                        + rcpt_qty [cv(location), cv(week) ],
sales_so_for [location, week] = sales_qty [cv(location), cv(week)] +
                        nvl(sales_so_for [cv(location), cv(week)-1],0),
rcpt_so_for [location, week] = rcpt_qty [cv(location), cv(week)] +
                        nvl(rcpt_so_for [cv(location), cv(week)-1],0)
)
order by item , location, week
/
```

Model

- Emulates spreadsheet like functionality
 - Very powerful, very flexible
 - Code is very compact
-

Model

Table used: item_Data , Data : Randomly generated

```
CREATE TABLE ITEM_DATA
```

```
(  
  ITEM          VARCHAR2(10 BYTE),  
  LOCATION      NUMBER,  
  WEEK          NUMBER,  
  SALES_QTY     NUMBER,  
  RCPT_QTY      NUMBER  
)
```

ITEM	LOCATION	WEEK	SALES_QTY	RCPT_QTY
Shirt	1	1	623	55
Shirt	1	2	250	469
Shirt	1	3	882	676
Shirt	1	4	614	856
Shirt	1	5	401	281
Shirt	1	6	163	581
shirt	1	7	324	415
Shirt	1	8	409	541
shirt	1	9	891	790
Shirt	1	10	759	465
Shirt	2	1	261	515
shirt	2	2	664	67

Model

Accesses previous row

- Let's say, we want to create an inventory report.

- Formula:

Inventory = last_week_inventory +
this week receipts -
this week sales

Model – Excel like..

	A	B	C	D	E						
1		week	----->								
2	location	1	2	3	4	5	6	7	8	9	10
3	1-sales_qty	199	433	888	165	338	611	858	37	778	140
4	1-rcpt_qty	232	787	714	836	245	699	489	843	787	674
5	1-inventory	33	387	213	884	791	879	510	1316	1325	1859
	2-sales_qty	697	263	...							
	2-rcpt_qty	676	112	...							

In Excel, formula would be $B5+C4-C3$ in Shirts sheet.

Model

```
select item, location, week, inventory, sales_qty, rcpt_qty
from item_data
model return updated rows
partition by (item)
dimension by (location, week)
measures ( 0 inventory , sales_qty, rcpt_qty)
rules (
inventory [location, week]= nvl(inventory[cv(location), cv(week)-
    1],0)
                                - sales_qty [cv(location), cv(week) ]
                                +rcpt_qty [cv(location), cv(week) ]
)
order by item , location, week
/
Script: model_example2.sql
```

Model

$$884 = 213 + 836 - 165$$

SQL Output:

ITEM	LOCATION	WEEK	INVENTORY	SALES_QTY	RCPT_QTY
Pants	1	1	33	199	232
Pants	1	2	387	433	787
Pants	1	3	213	888	714
Pants	1	4	884	165	836
Pants	1	5	791	338	245
Pants	1	6	879	611	699
Pants	1	7	510	858	489
Pants	1	8	1316	37	843
Pants	1	9	1325	778	787
Pants	1	10	1859	140	674
Pants	2	1	-21	697	676
Pants	2	2	-172	263	112

Notice the value is reset, at the onset of different location.

Return only changed rows

Model

```
select item, location, week, inventory, sales_qty, rcpt_qty  
from item_data
```

```
model return updated rows
```

```
partition by (item)
```

**Groups rows with
same item**

```
dimension by (location, week)
```

```
measures ( 0 inventory , sales_qty, rcpt_qty)
```

```
rules (
```

```
inventory [location, week]= nvl(inventory[cv(location), cv(week)-  
1],0)
```

```
- sales_qty [cv(location), cv(week) ]
```

```
+rcpt_qty [cv(location), cv(week) ]
```

```
)
```

```
order by item , location,week
```

```
/
```

Model

Equivalent to rows and columns in spreadsheet.

```
select item, location, week, inventory, sales_qty, rcpt_qty  
from item_data
```

```
model return updated rows
```

```
partition by (item)
```

```
dimension by (location, week)
```

```
{ measures ( 0 inventory , sales_qty, rcpt_qty) }
```

```
rules (
```

```
inventory [location, week]= nvl(inventory[cv(location), cv(week)-  
1],0)
```

```
- sales_qty [cv(location), cv(week) ]
```

```
+rcpt_qty [cv(location), cv(week) ]
```

```
)
```

```
order by item , location,week
```

```
/
```

Metrics that will be used in the formula

**inventory [1,3]= nvl(inventory[1, 2],0)
- sales_qty [1,3]
+rcpt_qty [1, 3)]**

Formula

Location	Week	Sales	Rcpt	Inventory
1	1	623	55	0+(623-55)=568
1	2	250	469	568+(250-469)=349
1	3	882	676	349+(882-676)=555
1	4	614	856	555+(614-856)=313
1	5	401	281	313+(401-281)=433

**inventory [location, week]= nvl(inventory[cv(location), cv(week)-
1],0)
- sales_qty [cv(location), cv(week)]
+rcpt_qty [cv(location), cv(week)]**

Model

CV(location) means, current value of location from the left side.

```
select item, location, week, inventory, sales_qty, rcpt_qty
from item_data
model return updated rows
partition by (item)
dimension by (location, week)
```

```
measures ( 0 inventory , sales_qty, rcpt_qty )
```

```
rules (
```

```
inventory [location, week]=
```

```
    nvl(inventory[cv(location), cv(week)-1],0)
```

```
    - sales_qty [cv(location), cv(week) ]
```

```
    + rcpt_qty [cv(location), cv(week) ]
```

```
)
```

```
order by item , location,week
```

```
/
```

Model

```
select item, location, week, inventory, sales_qty, rcpt_qty
from item_data
model return updated rows
partition by (item)
dimension by (location, week)
measures ( 0 inventory , sales_qty, rcpt_qty)
```

```
rules (
inventory [location, week]=      1      3      1      3
    nvl(inventory[cv(location), cv(week)-1],0)
    - sales_qty [cv(location), cv(week) ]
    + rcpt_qty [cv(location), cv(week) ]
)
```

```
order by item , location,week
/
```

<pre>inventory [1,3]= nvl(inventory[1, 2],0) - sales_qty [1,3] +rcpt_qty [1, 3)]</pre>
--

Model

SQL Output:

ITEM	LOCATION	WEEK	INVENTORY	SALES_QTY	RCPT_QTY
Pants	1	1	33	199	232
Pants	1	2	387	433	787
Pants	1	3	213	888	714
Pants	1	4	884	165	836
Pants	1	5	791	338	245
Pants	1	6	879	611	699
Pants	1	7	510	858	489
Pants	1	8	1316	37	843
Pants	1	9	1325	778	787
Pants	1	10	1859	140	674
Pants	2	1	-21	697	676
Pants	2	2	-172	263	112

Model

Sales_so_for is a running total of sales for this location so far..

- Of course complex queries possible:

```
select item, location, week, inventory, sales_qty, rcpt_qty , sales_so_for, rcpt_so_for
  from item_data
  model return updated rows
  partition by (item)
  dimension by (location, week)
  measures ( 0 inventory , sales_qty, rcpt_qty, 0 sales_so_for, 0 rcpt_so_for )
  rules (
    inventory [location, week] = nvl(inventory [cv(location), cv(week)-1 ] ,0)
      - sales_qty [cv(location), cv(week) ] +
      + rcpt_qty [cv(location), cv(week) ],
    sales_so_for [location, week] = sales_qty [cv(location), cv(week)] +
      nvl(sales_so_for [cv(location), cv(week)-1],0),
    rcpt_so_for [location, week] = rcpt_qty [cv(location), cv(week)] +
      nvl(rcpt_so_for [cv(location), cv(week)-1],0)
  )
  order by item , location, week
```

Script: model_example3.sql

Model - Rules

sales_so_for [location, week] =
sales_qty [cv(location), cv(week)] +
nvl(sales_so_for [cv(location), cv(week)-1],0),

sales_so_for [10, 1] = sales_qty [10, 1] +
nvl (sales_so_for [10, 0], 0)

Model - rules

```
rcpt_so_for [location, week] =  
    rcpt_qty [cv(location), cv(week)] +  
    nvl(rcpt_so_for [cv(location), cv(week)-1],0)
```

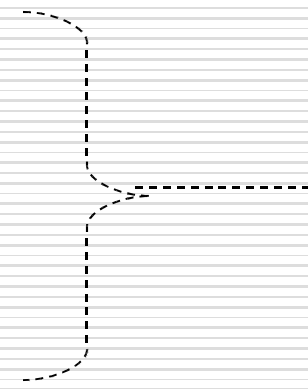
```
rcpt_so_for [2, 4] =  
    rcpt_qty [2, 4] +  
    nvl(rcpt_so_for [2, 4-1],0)
```

Model

Calculating the moving sales avg using this week and past two weeks..

- This SQL is finding the moving average

```
select item, location, week, sales_qty, rcpt_qty , moving_sales_avg
  from item_data
  model return updated rows
  partition by (item)
  dimension by (location, week)
  measures ( 0 moving_sales_avg , sales_qty, rcpt_qty)
  rules (
    moving_sales_avg [location, week] =
    (sales_qty [cv(location), cv(week)]+
      nvl(
        sales_qty [cv(location), cv(week)-1 ],
        sales_qty [cv(location), cv(week)]
      ) +
      nvl(
        sales_qty [cv(location), cv(week)-2 ],
        sales_qty [cv(location), cv(week)]
      )
    ) /3
  )
  order by item , location, week
Script : model_example4.sql
```



Model – explain plan

- As expected, explain plan has new keyword for model SQLs.

Rows	Row Source Operation
1500	SORT ORDER BY (cr=9 pr=0 pw=0 time=156936 us)
1500	SQL <u>MODEL</u> ORDERED (cr=9 pr=0 pw=0 time=102891 us)
1500	TABLE ACCESS FULL OBJ#(29061) (cr=9 pr=0 pw=0 time=85 us)

Analytic functions

Analytic functions

- Powerful feature provides ability to access previous and following rows
 - SQL can be written concisely using these functions
 - Works great for data warehouse and reporting queries
-

To calculate running total of sales for an item and location (store) by week.

Running sales total

```
select item, location, week, sales_qty, rcpt_qty,  
       sum (sales_qty)  
         over (partition by item, location  
              order by week  
              rows between  
                    unbounded preceding and current row  
                ) running_sales_total  
from item_data
```

Script: anfn_example1.sql

Running sales total

ITEM	LOCATION	WEEK	SALES_QTY	RCPT_QTY	RUNNING_SALES_TOTAL
Pants	1	1	638	524	638
Pants	1	2	709	353	1347
Pants	1	3	775	734	2122
Pants	1	4	344	879	2466
Pants	1	5	990	744	3456
Pants	1	6	635	409	4091
Pants	1	7	524	801	4615
Pants	1	8	14	593	4629
Pants	1	9	630	646	5259
Pants	1	10	61	241	5320
Pants	2	1	315	737	315
Pants	2	2	176	282	491
Pants	2	3	999	314	1490

This is considered a data partition

Running sales total

ITEM	LOCATION	WEEK	SALES_QTY	RCPT_QTY	RUNNING_SALES_TOTAL
Pants	1	1	638	524	638
Pants	1	2	709	353	1347
Pants	1	3	775	734	2122
Pants	1	4	344	879	2466
Pants	1	5	990	744	3456
Pants	1	6	635	409	4091
Pants	1	7	524	801	4615
Pants	1	8	14	593	4629
Pants	1	9	630	646	5259
Pants	1	10	61	241	5320
Pants	2	1	315	737	315
Pants	2	2	176	282	491
Pants	2	3	999	314	1490

Running sales total

```
select item, location, week, sales_qty, rcpt_qty,  
       sum (sales_qty)  
       over (partition by item, location  
            order by week  
            rows between  
                unbounded preceding and current row  
            ) running_sales_total  
from item_data
```

Groups the data in to partitions.

**Sorts the data by week
Column within that partition**

Provides a range for the sum function to operate

Moving average

- You can also calculate the moving averages easily

```
select item, location, week, rcpt_qty, sales_qty,  
       avg(sales_qty) over (  
         partition by item, location  
         order by week  
         rows between 2 preceding and current row  
       ) moving_sales_avg  
from item_data
```

Moving average between current row and prior two rows



Script: anfn_example2.sql

Moving average

ITEM	LOCATION	WEEK	RCPT_QTY	SALES_QTY	MOVING_AVG
Pants	1	1	524	638	638.00
Pants	1	2	353	709	673.50
Pants	1	3	734	775	707.33
Pants	1	4	879	344	609.33
Pants	1	5	744	990	703.00
Pants	1	6	409	635	656.33
Pants	1	7	801	524	716.33
Pants	1	8	593	14	391.00
Pants	1	9	646	630	389.33
Pants	1	10	241	61	235.00
Pants	2	1	737	315	315.00
Pants	2	2	282	176	245.50
Pants	2	3	314	999	496.67

Average

Lead

- ❑ Lead function provides ability to access later rows from that data partition
 - ❑ You can even specify the offset and default values.
-

Lead

Physical offset

Default value if the offset is beyond last row

```
select
  item, week, location , sales_qty,
  lead (sales_qty, 2, sales_qty ) over (
    partition by item, week order by sales_qty desc ) sls_qty
from item_data
/
```

ITEM	WEEK	LOCATION	SALES_QTY	SLS_QTY
Pants	1	43	990	957
Pants	1	10	976	941
Pants	1	31	957	941
Pants	1	5	941	888
Pants	1	45	941	874
Pants	1	44	888	843
Pants	1	27	874	836
Pants	1	14	843	832
Pants	1	42	836	827

Lag function

- ❑ Lag function is used to access values from the previous rows, in that data partition
 - ❑ If this is the first row in that partition, returns null.
-

Sales change percent formula=
 $((\text{this_week} - \text{last_week}) / \text{last_week}) * 100$

Lag function

```
select item, location, week, sales_qty, rcpt_qty ,  
       trunc (  
         (sales_qty - sales_prior_week) / sales_prior_week, 4) * 100  
         sales_change_percent  
from (  
select item, location, week, sales_qty, rcpt_qty,  
       lag(sales_qty) over (  
         partition by item, location  
         order by week  
         ) sales_prior_week  
from item_data  
)
```

**Lag function accesses
the sales_qty from
previous row.**

Sales change %

ITEM	LOCATION	WEEK	SALES_QTY	RCPT_QTY	SALES_CHANGE_PERCENT
Pants	1	1	638	524	
Pants	1	2	709	353	11.12
Pants	1	3	775	734	9.30
Pants	1	4	344	879	-55.61
Pants	1	5	990	744	187.79
Pants	1	6	635	409	-35.85
Pants	1	7	524	801	-17.48
Pants	1	8	14	593	-97.32
Pants	1	9	630	646	4400.00
Pants	1	10	61	241	-90.31
Pants	2	1	315	737	
Pants	2	2	176	282	-44.12
Pants	2	3	999	314	467.61

Lag and lead

Accesses the sales_qty from preceding week 2

```
select
  item, week, location ,
  lag (sales_qty, 2, sales_qty) over (
    partition by item, location order by week asc )
  prec_week_2_sls_qty,
  lag (sales_qty, 1, sales_qty ) over (
    partition by item, location order by week asc )
  prec_week_1_sls_qty,
  sales_qty,
  lead (sales_qty, 1, sales_qty ) over (
    partition by item, location order by week asc )
  succ_week_1_sls_qty,
  lead (sales_qty, 2, sales_qty ) over (
    partition by item, location order by week asc )
  succ_week_2_sls_qty
from item_data
```

Script: anfn_lead.sql

Lag and lead

This slide shows how we can access values from different rows easily, without any self-join.

ITEM	LOCATION	WEEK	Preceding week 2	Preceding week 1	This week	Succeeding week 1	Succeeding week 2
Pants	1	1	250	250	250	634	747
Pants	1	2	634	250	634	747	777
Pants	1	3	250	634	747	777	379
Pants	1	4	634	747	777	379	309
Pants	1	5	747	777	379	309	387
Pants	1	6	777	379	309	387	199
Pants	1	7	379	309	387	199	465
Pants	1	8	309	387	199	465	859
Pants	1	9	387	199	465	859	465
Pants	1	10	199	465	859	859	859
Pants	2	1	453	453	453	281	855
Pants	2	2	281	453	281	855	764
Pants	2	3	453	281	855	764	864

First_value/last_value

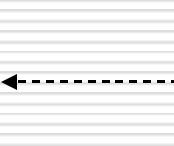
- These functions provides first or last values in an ordered set of rows
 - Very useful in finding the top value
-

First_value

```
select item, week, location, sales_qty
  from item_data
 order by item, week, location, sales_qty desc
 /
```

ITEM	WEEK	LOCATION	SALES_QTY
Pants	1	1	253
Pants	1	2	25
Pants	1	3	238
Pants	1	4	190
Pants	1	5	315
Pants	1	6	800
Pants	1	7	960
Pants	1	8	214
Pants	1	9	187
Pants	1	10	989
Pants	1	11	914
Pants	1	12	18
Pants	1	13	489
Pants	1	14	473

First_value should return
this row



First_value

```
select distinct item, week,  
  first_value(location) over (  
    partition by item, week  
    order by sales_qty desc  
    rows between unbounded preceding and unbounded following  
  ) top_seller,  
  first_value(sales_qty) over (  
    partition by item, week  
    order by sales_qty desc  
    rows between unbounded preceding and unbounded following  
  ) top_seller_qty  
from item_data  
order by item, week
```

ITEM	WEEK	TOP_SELLER	TOP_SELLER_QTY
Pants	1	10	989
Pants	2	38	997
Pants	3	22	997
Pants	4	29	987
...			

Retrieves the value of location, associated with the first row in that set

Partitions data by item, week and sorts them by sales_qty

First_value/last_value

```
select distinct item, week,  
first_value (location) over ( partition by item, week order by sales_qty desc  
rows between unbounded preceding and unbounded following ) top_seller,  
first_value (sales_qty) over (partition by item, week order by sales_qty desc  
rows between unbounded preceding and unbounded following )  
top_sales_qty,  
last_value (location) over (partition by item, week order by sales_qty desc  
rows between unbounded preceding and unbounded following )  
worst_seller,  
last_value (sales_qty) over (partition by item, week order by sales_qty desc  
rows between unbounded preceding and unbounded following )  
worst_sales_qty  
from item_data  
/
```

Script : anfn_first_value.sql

Top sellers by week

First_value/last_value

ITEM	WEEK	TOP_SELLER	TOP_SALES_QTY	WORST_SELLER	WORST_SALES_QTY
Pants	1	43	990	49	39
Pants	2	50	980	28	10
Pants	3	2	999	50	43
Pants	4	7	988	44	81
Pants	5	1	990	40	11
Pants	6	42	982	47	37
Pants	7	39	977	14	18
Pants	8	36	995	1	14
Pants	9	26	981	21	30

First_value-old way

```
select mi.item, mi.week, mi.min_sales_qty, mi.min_location,
ma.max_sales_qty, ma.max_location
from
(
select item, week, sales_qty max_sales_qty, location max_location from item_data
where (item, week, sales_qty) in
(
select item, week, max(sales_qty) from
item_data
group by item, week
)
) ma,
(
select item, week, sales_qty min_sales_qty, location min_location from item_data
where (item, week, sales_qty ) in
(
select item, week, min(sales_qty) from
item_data
group by item, week
)
) mi
where mi.item = ma.item and
mi.week = ma.week
order by item, week
/
```

Item_data accessed four times.
9 buffer gets using analytic function
and 36 buffer gets old way

Dense_rank

- Dense_rank provides rank of a row in a data partition.

```
select distinct item,week, location , sales_qty,  
dense_rank () over ( partition by item, week  
                    order by sales_qty desc ) sls_rnk  
from item_data  
order by item, week,sls_rnk  
/
```

ITEM	WEEK	LOCATION	SALES_QTY	SLS_RNK
Pants	1	35	984	1
Pants	1	26	925	2
Pants	1	39	922	3
Pants	1	40	897	4
Pants	1	34	889	5
Pants	1	7	870	6
Pants	1	14	866	7

Find top two stores per week in sales_qty and their rank in receipt_qty

Dense_rank

- Find top two stores

```
select * from (  
  select  
    item, week, location , sales_qty,  
    dense_rank () over ( partition by item, week  
                        order by sales_qty desc ) sls_rnk,  
    dense_rank () over ( partition by item, week  
                        order by rcpt_qty desc ) rcpt_rnk  
  from item_data  
 ) where sls_rnk < 3  
/
```

Ranks the row by rcpt_qty desc

Selecting top 2 rows by sls_rnk column

Dense_rank

ITEM	WEEK	LOCATION	SALES_QTY	SLS_RNK	RCPT_RNK
Pants	1	43	990	1	34
Pants	1	10	976	2	31
Pants	2	50	980	1	7
Pants	2	12	969	2	14
Pants	2	40	969	2	35
Pants	3	2	999	1	37
Pants	3	30	965	2	12
Pants	4	7	988	1	43
Pants	4	9	986	2	48
Pants	5	1	990	1	10
Pants	5	45	977	2	45
Pants	6	42	982	1	11
Pants	6	2	978	2	1
Pants	7	39	977	1	4
Pants	7	44	975	2	43
Pants	8	36	995	1	4
Pants	8	43	980	2	34
Pants	9	26	981	1	6
Pants	9	33	918	2	9
Pants	10	42	940	1	20
Pants	10	40	907	2	39
Shirt	1	21	980	1	17
Shirt	1	18	977	2	43
Shirt	2	39	918	1	44

Script : anfn_dense_rank.sql

Ntile

- Ntile function divides ordered set of rows in to buckets
- Each row is assigned a bucket

Script: `anfn_ntile.sql`

Divides the values in to 10 different buckets ordered by total_sls_qty

Ntile

```
select item, location, total_sls_qty, ntile(10) over ( partition by item order
    by total_sls_qty desc ) sales_group
from
    (select item, location, sum(sales_qty) total_sls_qty from item_data
    group by item, location)
```

/

ITEM	LOCATION	TOTAL_SLS_QTY	SALES_GROUP
Pants	45	6924	1
Pants	36	6725	1
Pants	33	6662	1
Pants	42	6280	1
Pants	15	6149	1
Pants	3	6106	2
Pants	24	6085	2
Pants	43	6061	2
Pants	30	6059	2
Pants	7	6002	2
Pants	12	5931	3
Pants	4	5889	3
Pants	26	5863	3
Pants	31	5754	3
....			

Top sellers

Show only the group of stores which are in the middle range. These stores have higher potential for improvement.

Ntile

```
select * from (  
select item, location, total_sls_qty, ntile(10) over ( partition by item order by total_sls_qty desc )  
      sales_group  
from  
(select item, location, sum(sales_qty) total_sls_qty from item_data  
  group by item, location)  
)  
where sales_group in (5,6)  
/
```

ITEM	LOCATION	TOTAL_SLS_QTY	SALES_GROUP
Pants	10	5319	5
Pants	46	5229	5
Pants	48	5192	5
Pants	47	5146	5
Pants	27	5098	5
Pants	40	5080	6
Pants	50	5050	6
Pants	5	5030	6
Pants	44	4948	6
Pants	38	4844	6
Shirt	38	5296	5
Shirt	11	5260	5

Script: anfn_ntile.sql

Regular expressions

Regular expressions

- Very effective in searching based upon a pattern.

Examples:

- Perform case insensitive search to find a product which has 'blue' in its name
 - Search to find names of the books with repeat strings
-

Regular expressions

- Four functions available
 - Regexp_like
 - Regexp_substr
 - Regexp_instr
 - Regexp_replace
-

Regular expressions

- Let's consider this table:

```
select name from book_master  
SQL> /
```

NAME

The Very Hungry Caterpillar board book
Brown Bear, Brown Bear, What Do You See?
Panda Bear, Panda Bear, What Do You See?
Cost-Based Oracle Fundamentals
Oracle Wait Interface

Regular expressions

Problem: Search for books with repeated patterns in their name

~~The Very Hungry Caterpillar board book~~

Brown Bear, Brown Bear, What Do You See?

Panda Bear, Panda Bear, What Do You See?

~~Cost-Based Oracle Fundamentals~~

~~Oracle Wait Interface~~

Regular expressions

Problem: Search for book names with patterns repeated

```
select
regexp_substr (name, '([[:alnum:]]|[:space:]){10,}).*(\1)') sbstr_out,
regexp_instr (name, '([[:alnum:]]|[:space:]){10,}).*(\1)') instr_out
from book_master
where
regexp_like (name, '([[:alnum:]]|[:space:]){10,}).*(\1)')
/
```

SBSTR_OUT	INSTR_OUT
-----	-----
Brown Bear, Brown Bear	1
Panda Bear, Panda Bear	1

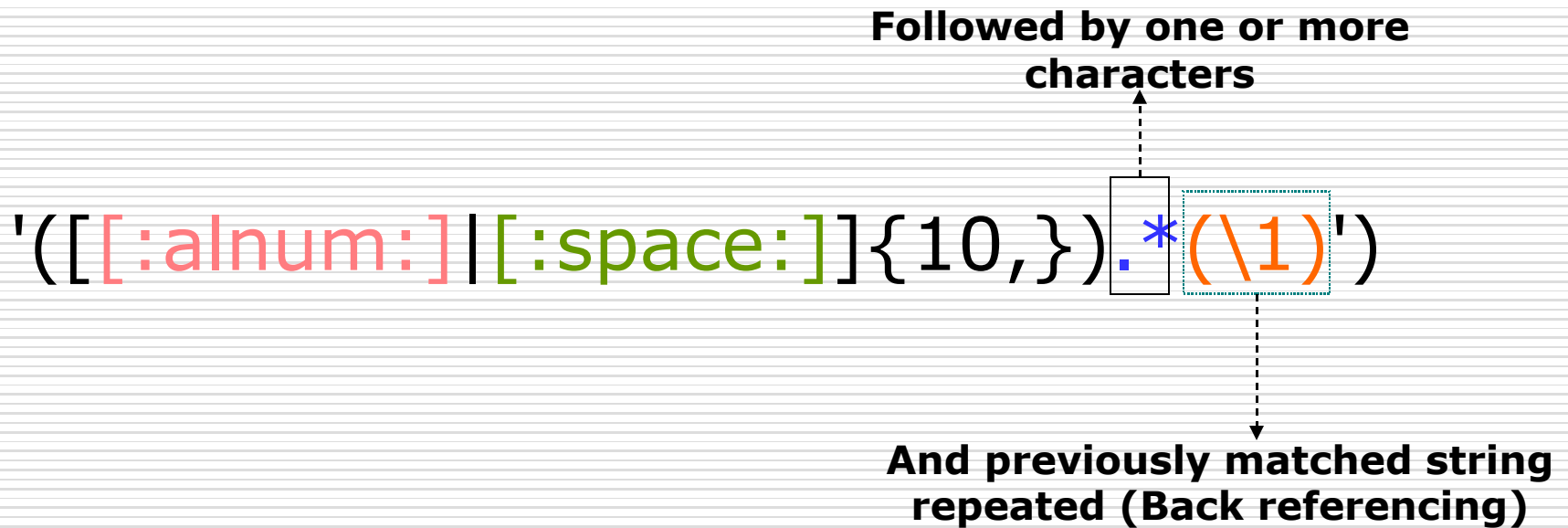
Regular expressions

One or more words with at least 10 alphanumeric characters and space.

'([[:alnum:]]|[:space:]){10,}.*(\1)'

Space
Alphanumeric characters

Regular expressions



Regular expressions

One or more words with at least 10 alphanumeric characters and space.

Followed by one or more characters

'([[:alnum:]]|[:space:]){10,}.*(\1)'

And previously matched string repeated (Back referencing)

Brown bear ,

Brown bear

Regular expressions

- Regexp_replace is useful in creating new strings, based upon existing string.

```
select name from class_roster;  
NAME
```

```
George :: Adams  
Scott  :: Davey  
Brian  :: Williams
```

Format of name is First_name:: Last_name

Script: regexp_02.sql

Regular expressions

Problem : Convert strings of the format
First_name:: Last_name to
Last_name, first_name

George :: Adams to Adams , George

Regular expressions

- Solution for previous problem is:

```
Select name,  
regexp_replace(name,  
'([[:alnum:]]|[:space:]){1,})::([[:alnum:]]|[:space:]){1,})' ,  
    '\2,\1' ) last_comma_first  
from class_roster  
/
```

Regular expressions

`([[:alnum:]]|[:space:]){1,}`

`([[:alnum:]]|[:space:]){1,}`

One or more characters of
alphanumeric or space

One or more characters of
alphanumeric or space

`\1`

`\2`

`\2,\1`

Regular expressions

- Output of the query is:

NAME	LAST_COMMA_FIRST
George :: Adams	Adams, George
Scott :: Davey	Davey, Scott
Brian :: Williams	Williams, Brian

Flashback feature

Flashback features

- Many different variations of flashback feature:
 - Flashback query *
 - Flashback transaction query*
 - Flashback table*
 - Flashback drop
 - Flashback database

* Only these three are considered here

Flashback versions query

- We can query the table row as of a point in time
 - This feature uses undo segments.
 - Undo_retention parameter plays a key role.
-

Flashback query

□ Table emp_salary

```
create table emp_salary  
  ( emp_id number,  
    salary number  
  );
```

```
insert into emp_salary (emp_id, salary) values ( 1, 2000);  
insert into emp_salary (emp_id, salary) values ( 2, 3000);  
insert into emp_salary (emp_id, salary) values ( 3, 4000);  
insert into emp_salary (emp_id, salary) values ( 4, 5000);  
insert into emp_salary (emp_id, salary) values ( 5, 6000);
```

Script: flashback_version_query.sql

Flashback query

-- Initial version of the row

```
select emp_id, salary from emp_salary where emp_id=5;
```

EMP_ID	SALARY
5	6000

```
update emp_salary set salary=salary*1.25 where emp_id=5 ;  
Commit;
```

```
select emp_id, salary from emp_salary where emp_id=5;
```

EMP_ID	SALARY
5	7500

```
update emp_salary set salary=salary*1.25 where emp_id=5 ;  
Commit;
```

```
select emp_id, salary from emp_salary where emp_id=5;
```

EMP_ID	SALARY
5	9375

Flashback query

- We realized that salary is updated for that row. We want to see all versions of the row:

```
select emp_id, salary, versions_xid, versions_starttime,  
       versions_endtime
```

```
from emp_salary
```

versions between scn minvalue and maxvalue

```
where emp_id=5 order by versions_endtime nulls last;
```

EMP_ID	SALARY	VERSIONS_XID	VERSIONS_STARTTIME	VERSIONS_ENDTIME
5	6000			28-JUL-06 04.21.53 PM
5	7500	0001002200001E32	28-JUL-06 04.21.53 PM	28-JUL-06 04.21.53 PM
5	9375	0003001F00003526	28-JUL-06 04.21.53 PM	

- Current version of the row
- New version of the row started and ended
- Initial version of row

Flashback query

- Another variation of flashback query is:

```
select emp_id, salary
from emp_salary
as of timestamp systimestamp-(5/(24*60*60))
where emp_id=5
/
```

EMP_ID	SALARY
5	6000

Row version as of 5 seconds ago.

Flashback query

- You could also use scn based query

```
select emp_id, salary
from emp_salary
as of scn 5615683784747
where emp_id=5;
```

EMP_ID	SALARY
5	6000

Flashback TX query

- Activity from a specific transaction can be queried too:

```
select start_timestamp , logon_user, undo_sql
from flashback_transaction_query
where xid=hextoraw('0001000E00001EA6' )
;
```

```
START_TIM LOGON_USER
```

```
-----
UNDO_SQL
```

```
02-AUG-06 SQLNF
```

```
update "SQLNF"."EMP_SALARY" set "SALARY" = '5000' where ROWID =
'AAAH5aAAEAAACgSAAD';
```

```
02-AUG-06 SQLNF
```

```
update "SQLNF"."EMP_SALARY" set "SALARY" = '7500' where ROWID =
'AAAH5aAAEAAACgSAAE';
```

```
script: flashback_version_tx.sql
```



Versions_xid value
queried from prior
SQL

Flashback table

- ❑ Flashback table command can be used to recover the table to a point in time.
 - ❑ Depends very much on undo_Retention and undo activity.
 - ❑ Not a substitute for backup.
-

Flashback table

Initial version of rows..

EMP_ID	SALARY
1	2000
2	3000
3	4000
4	5000
5	6000

We will update the table adding 25% to emp_id in (4,5).

Another transaction to update a row again. COMMIT;

Modified versions of row

EMP_ID	SALARY
1	2000
2	3000
3	4000
4	6250
5	9375

Script : flashback_version_table.sql

Flashback table

- Now, we can use the flashback table command to recover the table to a prior scn or time.

```
flashback table emp_salary to scn 5615683871107;
```

```
select * from emp_salary;
```

EMP_ID	SALARY
1	2000
2	3000
3	4000
4	5000
5	6000

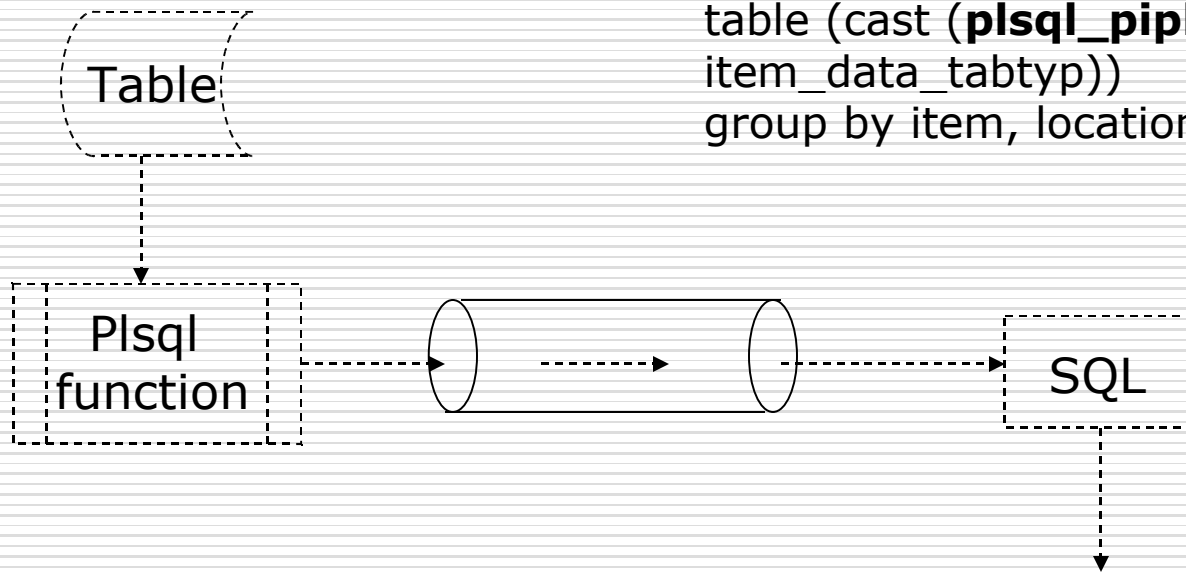
PL/SQL pipelined functions

Pipelined functions

- ❑ User defined functions can be accessed as if the function is a table.
 - ❑ Useful in data warehouse applications where data potentially goes through multiple transformations.
 - ❑ Rows can be returned iteratively.
-

Pipelined functions

```
select item, location, sum(qty) from  
table (cast (plsql_piplined_noparallel as  
item_data_tabtyp))  
group by item, location;
```



ITEM	LOCATION	SUM(QTY)
Pants	1	1091215.73
Pants	13	700517.6

Pipelined functions

- Let's define few types..

REM Type of row returned.

```
create or replace type item_data_typ
as object
```

```
( item varchar2(30),
  location number,
  qty number
```

```
)
/
```

REM table of above type

```
create or replace type item_data_tabtyp as table of
  item_data_typ
```

```
/
```

Pipelined functions

- We will define a pipelined function..

create or replace function plsqli_piplined_noparallel

return item_data_tabtyp **PIPELINED**

as

```
l_item_data_tabtyp item_data_tabtyp := item_data_tabtyp ();
```

```
indx number:=1;
```

```
l_fib number :=0;
```

```
begin
```

```
l_item_data_tabtyp.extend;
```

```
for l_csr_items in (select item, location, qty from item_data )
```

```
loop
```

```
l_fib := next_Fib_N(round(l_csr_items.location) );
```

```
l_item_data_tabtyp (indx) := item_data_ttyp ( l_csr_items.item, l_fib, l_csr_items.qty );
```

```
pipe row ( l_item_data_tabtyp (indx) );
```

```
l_item_data_tabtyp.extend;
```

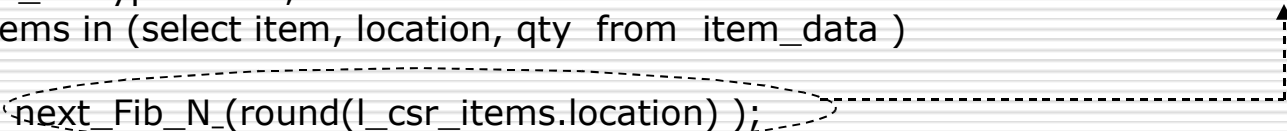
```
indx := indx+1;
```

```
end loop;
```

```
return;
```

```
end;
```

Calling an
hypothetical
transformation
function : A
fibonacci number



Pipelined functions

REM Now, call the pipelined function and cast it to item_data_tabtyp
REM object type..

```
select item, location, sum(qty) from  
table (cast (plsql_piplined_noparallel as item_data_tabtyp))  
group by item, location
```

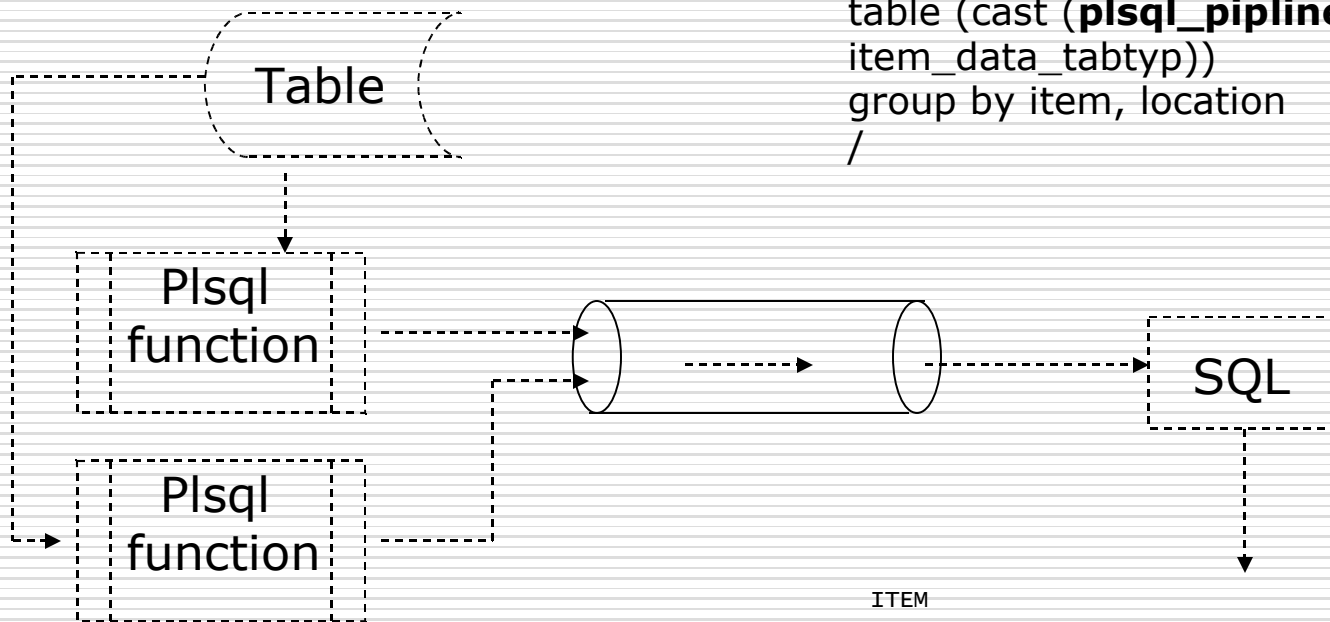
```
/
```

ITEM	LOCATION	SUM(QTY)
Pants	1	1091215.73
Pants	13	700517.6
Pants	21	763532.279
Pants	34	706139.503
Pants	55	1495631.24
Pants	89	1534359.5

```
Script : plsql_populate_table.sql  
        plsql_table_fnc_01.sql
```

Pipelined parallel functions

```
select item, location, sum(qty) from  
table (cast (plsql_piplined_parallel as  
item_data_tabtyp))  
group by item, location  
/
```



ITEM	LOCATION	SUM(QTY)
Pants	1	1091215.73
Pants	13	700517.6

Pipelined functions

```
create or replace function plsqli_piplined_parallel (
    l_item_data_ref in item_data_pkg.item_data_ref )
return item_data_tabtyp PIPELINED
parallel_enable (partition l_item_data_ref by range (location) )
as
    l_item_data_tabtyp item_data_tabtyp := item_data_tabtyp ();
    indx number:=1;
    l_fib number :=0;
    l_item varchar2(30);
    l_location number;
    l_qty number;

    l_sid number;
    l_dummy_sid number;
```

<... continued..>

Pipelined functions

```
...
Begin
  l_item_data_tabtyp.extend;

  select sid into l_sid from v$mystat where rownum <2;
  loop
    fetch l_item_data_ref into l_item , l_location, l_qty , l_dummy_sid ;
    exit when l_item_data_ref%NOTFOUND;
    l_fib := next_Fib_N (round(l_location) );
    l_item_data_tabtyp (indx) := item_data_typ ( l_item, l_fib, l_qty ,l_sid
);
    pipe row ( l_item_data_tabtyp (indx) );
    l_item_data_tabtyp.extend;
    indx := indx+1;
  end loop;
  return;
end;
/
```

Pipelined functions

```
select /*+ parallel (5) */ item, location ,sid, sum(qty) from
  table (
    cast (
      plsql_piplied_parallel(cursor (select item, location, qty ,0 from item_data) )
      as item_data_tabtyp
    )
  )
group by item, location, sid
order by sid
```

ITEM	LOCATION	SID	SUM(QTY)
...			
Scarves	987	182	5226861.79
Pants	987	182	5243932.25
Pants	21	182	763532.279
shirts	55	195	1507026.7
shirts	2584	195	8130601.01
shirts	610	195	3692315.45
...			

Rows
processed
by sid 182

Rows
processed
by sid 195

Script: plsql_table_fnc_parallel.sql

PL/SQL warnings

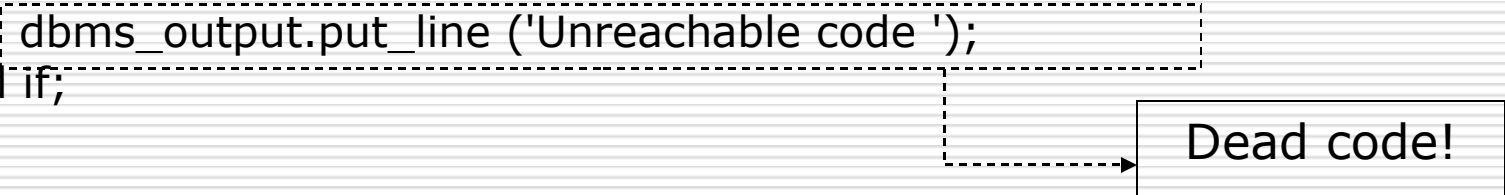
PL/SQL warnings

- ❑ Oracle 10g introduces PL/SQL warnings.
- ❑ While creating a procedure warnings can be printed.
- ❑ This can be enabled at session level.

```
alter session set plsql_warnings='enable:all';
```

PL/SQL warnings

```
alter session set plsql_warnings='enable:all';
drop procedure plsql_warnings_prc_01;
create or replace procedure plsql_warnings_prc_01
is
  l_id number :=1;
begin
  if (l_id =1) then
    dbms_output.put_line ('Value of l_id is '|| l_id);
  else
    dbms_output.put_line ('Unreachable code ');
  end if;
end;
```



Show errors

```
Errors for PROCEDURE PLSQL_WARNINGS_PRC_01:
LINE/COL ERROR
```

9/2 PLW-06002: Unreachable code

Script : plsql_warnings_01.sql

PL/SQL warnings

- General syntax to modify this parameter is :

```
{ ENABLE | DISABLE | ERROR }:  
{ ALL| SEVERE| INFORMATIONAL| PERFORMANCE  
  | { integer| (integer [, integer ] ...)}  
}
```

E.g.: 'ENABLE:SEVERE','DISABLE:5000'

Refer \$ORACLE_HOME/plsql/mesg/plwus.mesg file for more information for the error codes.

PL/SQL warnings

- But, this seems to work only if the procedure is newly created.

```
*** Drop and recreate procedure ***  
Procedure dropped.
```

```
Procedure created.
```

```
SP2-0804: Procedure created with compilation warnings
```

```
Errors for PROCEDURE PLSQL_WARNINGS_PRC_01:  
LINE/COL ERROR
```

```
-----  
9/2    PLW-06002: Unreachable code
```

Creating the procedure first time prints these warning messages!

```
*** Recreating second time ***
```

```
Procedure created.
```

No messages printed
If the procedure is pre-existing.

PL/SQL warnings

```
alter session set plsql_warnings='enable:all';
```

```
drop procedure plsql_warnings_prc_02;
```

```
create or replace procedure plsql_warnings_prc_02
```

```
is  
  l_id number :=1;  
begin  
  for i in 1 .. 5  
  loop  
    l_id := 2;  
  end loop;  
end;  
/
```

This statement is inefficient and probably should be moved outside the loop!

```
Errors for PROCEDURE PLSQL_WARNINGS_PRC_02:
```

```
LINE/COL ERROR
```

```
-----  
7/2    PLW-06002: Unreachable code  
Script : plsql_warnings_02.sql
```

Message is nonsensical!

(or Is it ?)

PL/SQL warnings

```
alter session set plsql_warnings='enable:all';
```

```
drop procedure plsql_warnings_prc_02;  
create or replace procedure plsql_warnings_prc_02  
is  
  l_id number :=1;  
begin  
  for i in 1 .. 5  
  loop  
    l_id := 2;  
    dbms_output.put_line ('i ='||i);  
  end loop;  
end;  
/  
Procedure created.
```

Same code as previous screen, with additional dbms_output line.

Now, there is no warning, which is an expected behaviour.

```
show errors  
No errors.
```

PL/SQL warnings

prompt This procedure accepts clob as in out paramteter
prompt

```
create or replace procedure plsql_warnings_prc_05  
  ( l_emp_jpeg in out blob )
```

```
is
```

```
  l_id number := 1;
```

```
  l_lc number := 1;
```

```
begin
```

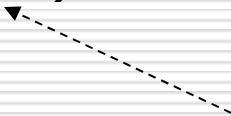
```
    l_id := 'ORCL';
```

```
    l_id := l_id + 1;
```

```
end;
```

```
/
```

```
Show errors
```



Potentially passing huge variable as a blob column. Let's see whether the engine can give us warnings.

PL/SQL warnings

prompt This procedure accepts clob as in out parameter

prompt

```
create or replace procedure plsql_warnings_prc_05
```

```
( l_emp_jpeg in out blob )
```

```
is
```

```
  l_id number :=1;
```

```
  l_lc number :=1;
```

```
begin
```

```
  l_id := 'ORCL';
```

```
  l_id := l_id + 1;
```

```
end;
```

```
/
```

Show errors

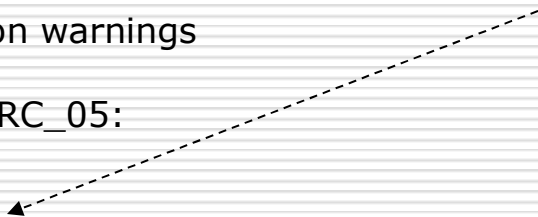
SP2-0804: Procedure created with compilation warnings

Errors for PROCEDURE PLSQL_WARNINGS_PRC_05:

LINE/COL ERROR

2/5 PLW-07203: parameter 'L_EMP_JPEG' may benefit from use of the NOCOPY compiler hint

PL/SQL engine correctly detected that this variable can benefit from NOCOPY hint. NOCOPY variables are passed by reference.



PL/SQL warnings

- Can this feature detect infinite loop in the code?

```
create or replace procedure plsql_warnings_prc_03
```

```
is
```

```
  l_id number := 1;
```

```
  l_lc number := 1;
```

```
begin
```

```
  while (l_lc <= 100) ←
```

Infinite loop

```
  loop
```

```
    l_id := l_id * l_id;
```

```
  end loop;
```

```
end;
```

PL/SQL warnings

- No errors in previous slide.
 - Detecting infinite loop is not a mundane task and it is known as 'halting problem'.

“[Alan Turing](#) proved in [1936](#) that a general [algorithm](#) to solve the halting problem for *all* possible program-input pairs cannot exist.” – Wiki
 - Hopefully, future release might detect simple infinite loops.
-

PL/SQL warnings

```
create or replace procedure plsql_warnings_prc_06
  ( l_emp_id in number )
is
  l_id number :=1;
  l_lc number :=1;
  cv number :=0;
begin
  l_id := 'ORCL';
  l_id := l_id + 1;
  cv :=1;
end;
/
```

CV is a keyword. Does this feature detect use of keywords ?

```
SP2-0804: Procedure created with compilation warnings
Errors for PROCEDURE PLSQL_WARNINGS_PRC_06:
LINE/COL ERROR
```

```
6/3      PLW-05004: identifier CV is also declared in STANDARD or is a SQL
        builtin
```

Concise warning message.

PL/SQL native compilation

PL/SQL native compilation

- PL/SQL is interpreted and code is converted to m-code.
 - For arithmetic intensive processing, natively compiled code is better for performance.
-

PL/SQL native compilation

- PL/SQL native compilation is possible in 9i and enhanced in 10g.
 - Native compilation converts PL/SQL in to C language and compiles them.
-

PL/SQL native compilation

-- Native compilation needs these three parameters set.

```
alter session set plsql_code_type=native;
```

```
alter session set plsql_debug=false;
```

```
alter system set plsql_native_library_dir='/tmp';
```

PL/SQL native compilation

```
create or replace procedure plsql_compile_prc_01
is
  l_id number :=1;
begin

  if (l_id =1) then
    dbms_output.put_line ('Value of l_id is '|| l_id);
  else
    dbms_output.put_line ('Unreachable code ');
  end if;
end;
/
```

Script: plsql_compile_01.sql

PL/SQL native compilation

```
/*----- Implementation of Procedure PLSQL_COMPILE_PRC_01 -----*/
```

```
# ifdef __cplusplus
extern "C" {
# endif
```

```
# ifndef PEN_ORACLE
# include <pen.h>
# endif
```

```
/* Types used in generated code */
```

```
typedef union {ub1 st[344]; size_t _si; void * _vs;} PEN_State;
typedef union {ub1 cup[328]; size_t _cu; void * _vc;} PEN_Cup;
typedef union {ub1 slg[112]; pen_buffer p;} PEN_Buffer;
```

```
/* Macros used in generated code */
```

```
#define dl0 ((void ***) (PEN_Registers[ 3]))
#define dpf ((void ****) (PEN_Registers[ 5]))
```

```
#define bit(x, y) ((x) & (y))
#define PETisstrnull(strhdl) \
    (!PMUflgnotnull(PETmut(strhdl)) || !PETdat(strhdl) || !PETlen(strhdl))
```

```
.....
```

Conclusion

- We covered many new features in SQL and PL/SQL
 - We have established that performance of many queries can be improved using these new features.
 - PL/SQL offers rich set of functionalities.
-

Thank you for listening.

QUESTIONS ?
