


Performance Management Using Statspack and Analytical SQL


Daniel Fink
OptimalDBA.com

1



- Daniel Fink
 - Senior Oracle Database Performance Engineer
 - Main focus on Diagnosis and Optimization
 - Data Recovery and Training
 - 11 years Oracle database administration
 - Member of the Oak Table Network

daniel.fink@optimaldba.com
www.optimaldba.com



Copyright 2007 OptimalDBA.com OptimalDBA.com 2

Agenda

- Overview of Statspack structures, snapshot process and basic logic in the context of gathering additional data
- Overview of Analytical SQL
- Demonstrate the usage of analytical functions on statspack data to present custom reports

Statspack

- Tables capture current values
- Amount of data captured is governed by
 - Snapshot level
 - Thresholds
- Supplied Report
 - Lots of data

Levels & Thresholds

- The higher the level, the more detail
 - 5 - best combination of detail/performance
 - 6 - execution plan detail
 - *Bug may cause it to hang querying v\$sql_plan
- Thresholds can be adjusted for each system
 - SQL thresholds are good for repeatable sql

Snapshots

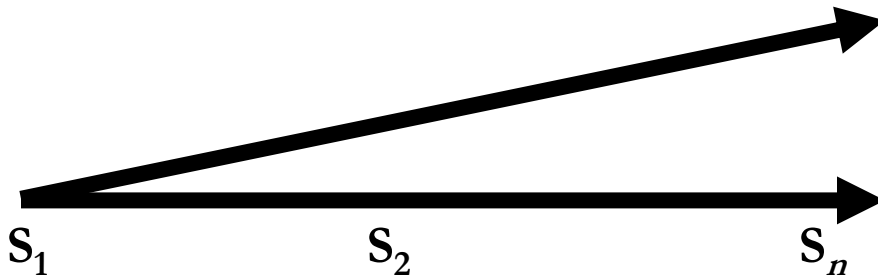
- Automated gathering
 - Avoiding snapshot creep
- Manual gathering
 - Use when problem is occurring **NOW**
- Use the comment field when you can

Statistics

- Statistics include
 - Event counts and time waited
 - System statistics
 - File statistics
- Statistics are monotonically increasing
 - Instance restart sets value of statistic at 0
 - The value of a statistic at instance shutdown cannot be known
 - Even with an on shutdown trigger
 - The 'final' value is the value at the last snapshot

Monotonically Increasing

- In the absence of an instance restart, the values for statistics and events will only increase.



Deltas

- To determine the actual value for a statistic
 - The starting point is the first snapshot
 - The ending point is the last snapshot
- If there is an instance restart between snapshots, intermediate deltas need to be calculated

How do we calculate deltas?

- Multiple passes on table
- DECODE/CASE
- PL/SQL
- Analytical SQL

Why Analytical SQL

- Improved code
 - Readability
 - Supportability
- Improved Performance
 - Fewer passes on a table
 - Less resource consumption
 - Faster run time
- Part of the core product

Employee	Salary
Thomas Walton	\$25,000
Eric Kraus	\$27,000
Larry Wilton	\$30,000
Allison Dietrich	\$30,000
Shelly Walton	\$30,000
Henry Parry	\$30,000
Billy Yolto	\$30,000
Bev George	\$30,000
William Dietrich	\$30,000
Bobby Harris	\$30,000
Oscar Perry	\$45,000
Stan Marsh	\$45,000
John Dennis	\$46,000
Vincent Johns	\$50,000
Doug Harris	\$50,000
Erika Deeter	\$60,000
Andy Schmidt	\$70,000
Julie Johnson	\$75,000
Tina Walton	\$80,000
Allison Ballinger	\$90,000
Rachel Middleton	\$850,000

```
SELECT e.name,  
       e.salary  
FROM   employee e  
ORDER BY e.salary
```

Employee	Salary
Rachel Middleton	\$850,000
Allison Ballinger	\$90,000
Tina Walton	\$80,000
Julie Johnson	\$75,000
Andy Schmidt	\$70,000
Erika Deeter	\$60,000
Doug Harris	\$50,000
Vincent Johns	\$50,000
John Dennis	\$46,000
Oscar Perry	\$45,000
Stan Marsh	\$45,000
Allison Dietrich	\$30,000
Shelly Walton	\$30,000
Henry Parry	\$30,000
Bev George	\$30,000
William Dietrich	\$30,000
Billy Yolto	\$30,000
Bobby Harris	\$30,000
Larry Wilton	\$30,000
Eric Kraus	\$27,000
Thomas Walton	\$25,000

```
SELECT e.name,
       e.salary
FROM   employee e
ORDER BY e.salary DESC
```

Rank

Employee	Salary	Rank
Rachel Middleton	\$850,000	1
Allison Ballinger	\$90,000	2
Tina Walton	\$80,000	3
Julie Johnson	\$75,000	4
Andy Schmidt	\$70,000	5
Erika Deeter	\$60,000	6
Doug Harris	\$50,000	7
Vincent Johns	\$50,000	7
John Dennis	\$46,000	9
Oscar Perry	\$45,000	10
Stan Marsh	\$45,000	10
Allison Dietrich	\$30,000	12
Shelly Walton	\$30,000	12
Henry Parry	\$30,000	12
Bev George	\$30,000	12
William Dietrich	\$30,000	12
Billy Yolto	\$30,000	12
Bobby Harris	\$30,000	12
Larry Wilton	\$30,000	12
Eric Kraus	\$27,000	20
Thomas Walton	\$25,000	21

```
SELECT e.name,
       e.salary,
       RANK()
         OVER (ORDER BY
              salary DESC)
          sal_rank
FROM   employee e
ORDER BY e.salary DESC
```

Employee	Salary	Rank
Rachel Middleton	\$850,000	1
Allison Ballinger	\$90,000	2
Tina Walton	\$80,000	3
Julie Johnson	\$75,000	4
Andy Schmidt	\$70,000	5
Erika Deeter	\$60,000	6
Doug Harris	\$50,000	7
Vincent Johns	\$50,000	7
John Dennis	\$46,000	9
Oscar Perry	\$45,000	10
Stan Marsh	\$45,000	10
Allison Dietrich	\$30,000	12
Shelly Walton	\$30,000	12
Henry Parry	\$30,000	12
Bev George	\$30,000	12
William Dietrich	\$30,000	12
Billy Yolto	\$30,000	12
Bobby Harris	\$30,000	12
Larry Wilton	\$30,000	12
Eric Kraus	\$27,000	20
Thomas Walton	\$25,000	21

```
SELECT e.name,
       e.salary,
       RANK()
         OVER (ORDER BY
              salary DESC)
         sal_rank
FROM   employee e
ORDER BY e.salary DESC
```

Dense Rank

Employee	Salary	Rank
Rachel Middleton	\$850,000	1
Allison Ballinger	\$90,000	2
Tina Walton	\$80,000	3
Julie Johnson	\$75,000	4
Andy Schmidt	\$70,000	5
Erika Deeter	\$60,000	6
Doug Harris	\$50,000	7
Vincent Johns	\$50,000	7
John Dennis	\$46,000	8
Oscar Perry	\$45,000	9
Stan Marsh	\$45,000	9
Allison Dietrich	\$30,000	10
Shelly Walton	\$30,000	10
Henry Parry	\$30,000	10
Bev George	\$30,000	10
William Dietrich	\$30,000	10
Billy Yolto	\$30,000	10
Bobby Harris	\$30,000	10
Larry Wilton	\$30,000	10
Eric Kraus	\$27,000	11
Thomas Walton	\$25,000	12

```
SELECT e.name,
       e.salary,
       DENSE_RANK()
         OVER (ORDER BY
              salary DESC)
         sal_rank
FROM   employee e
ORDER BY e.salary DESC
```

Rank, Dense Rank, Row Number

Employee	Salary	Rank	Dense	Row #
Rachel Middleton	\$850,000	1	1	1
Allison Ballinger	\$90,000	2	2	2
Tina Walton	\$80,000	3	3	3
Julie Johnson	\$75,000	4	4	4
Andy Schmidt	\$70,000	5	5	5
Erika Deeter	\$60,000	6	6	6
Doug Harris	\$50,000	7	7	7
Vincent Johns	\$50,000	7	7	8
John Dennis	\$46,000	9	8	9
Oscar Perry	\$45,000	10	9	10
Stan Marsh	\$45,000	10	9	11
Allison Dietrich	\$30,000	12	10	12
Shelly Walton	\$30,000	12	10	13
Henry Parry	\$30,000	12	10	14
Bev George	\$30,000	12	10	15
William Dietrich	\$30,000	12	10	16
Billy Yolto	\$30,000	12	10	17
Bobby Harris	\$30,000	12	10	18
Larry Wilton	\$30,000	12	10	19
Eric Kraus	\$27,000	20	11	20
Thomas Walton	\$25,000	21	12	21

```

SELECT      e.name,
            e.salary,
            DENSE_RANK()
              OVER (ORDER BY salary DESC)
              sal_rank
FROM        employee e
ORDER BY   e.salary DESC
    
```

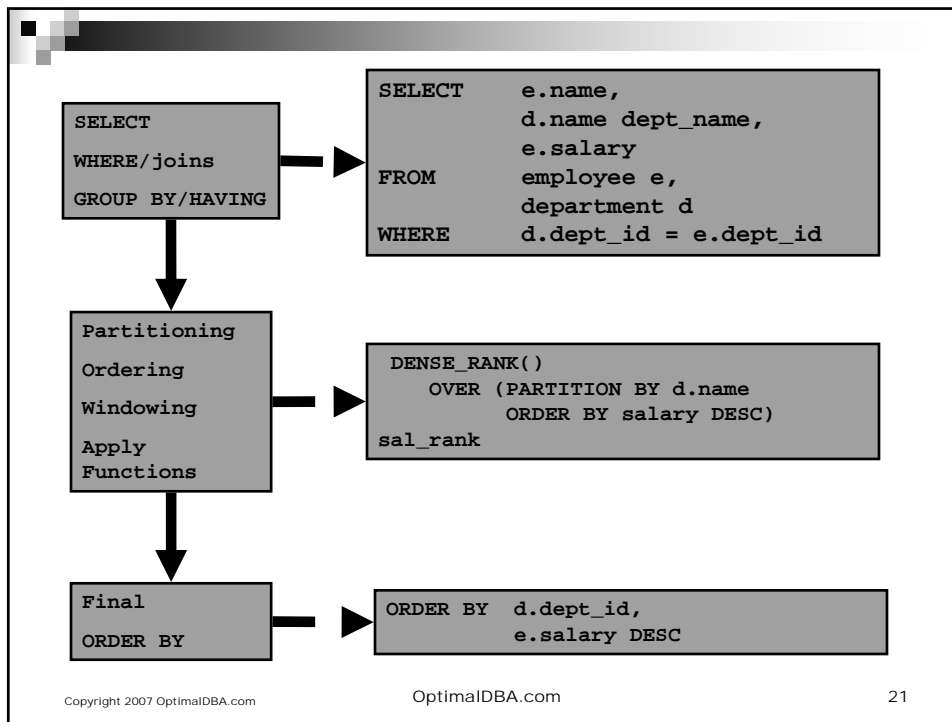
Analytical Function

ORDER BY clause

Partitioning

- *Not to be confused with table/index partitioning*
- A set of rows grouped by defined value or values
 - Default partition is entire result set
 - Functions are applied relative to the partition

```
SELECT      e.name,  
            d.name dept_name,  
            e.salary,  
            DENSE_RANK()  
              OVER (PARTITION BY dept_name,  
                    ORDER BY salary DESC) sal_rank  
FROM        employee e,  
            department d  
WHERE       d.dept_id = e.dept_id  
ORDER BY   d.dept_id, e.salary DESC
```



Filter limitation

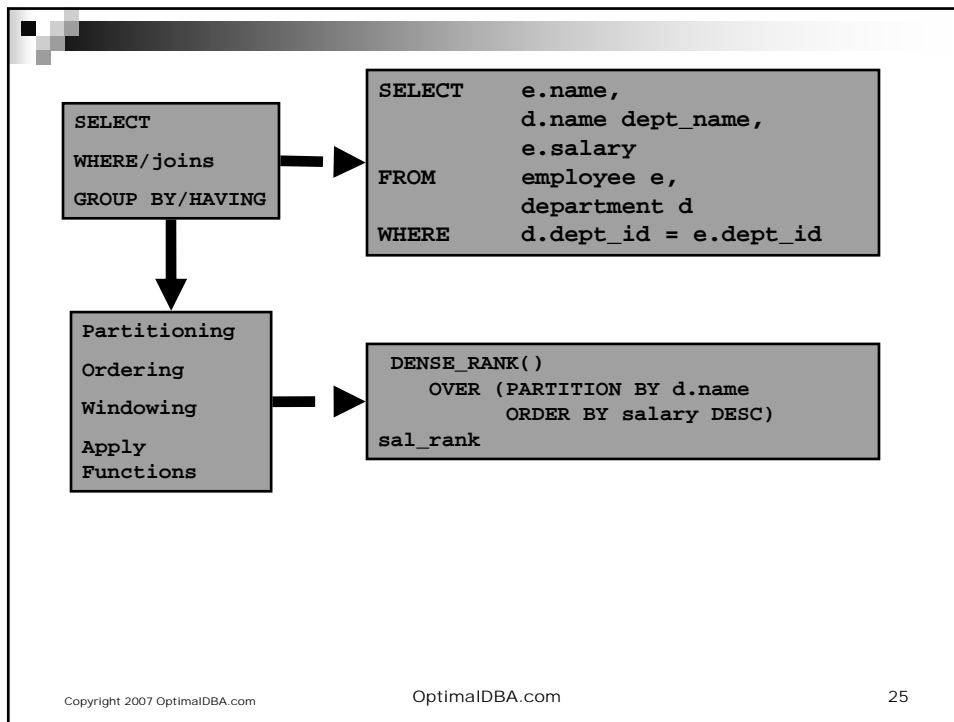
- Cannot be used to filter records
- WHERE (filter predicate) is applied BEFORE functions are processed

```
SELECT
WHERE/joins
GROUP BY/HAVING
```



```
SELECT    e.name,
          d.name dept_name,
          e.salary
FROM      employee e,
          department d
WHERE     d.dept_id = e.dept_id
```

Employee	Department	Salary
-----	-----	-----
Allison Ballinger	Administration	\$90,000
William Dietrich	Administration	\$30,000
Rachel Middleton	Administration	\$850,000
Julie Johnson	Sales and Marketing	\$75,000
Larry Wilton	Sales and Marketing	\$30,000
Shelly Walton	Sales and Marketing	\$30,000
Bobby Harris	Sales and Marketing	\$30,000
Allison Dietrich	Sales and Marketing	\$30,000
Andy Schmidt	Sales and Marketing	\$70,000
Henry Parry	Sales and Marketing	\$30,000
Bev George	Sales and Marketing	\$30,000
Oscar Perry	Sales and Marketing	\$45,000
Stan Marsh	Accounting	\$45,000
John Dennis	Accounting	\$46,000
Erika Deeter	Accounting	\$60,000
Thomas Walton	Logistics and Supply	\$25,000
Tina Walton	Logistics and Supply	\$80,000
Eric Kraus	Logistics and Supply	\$27,000
Doug Harris	Logistics and Supply	\$50,000
Billy Yolto	Logistics and Supply	\$30,000
Vincent Johns	Logistics and Supply	\$50,000

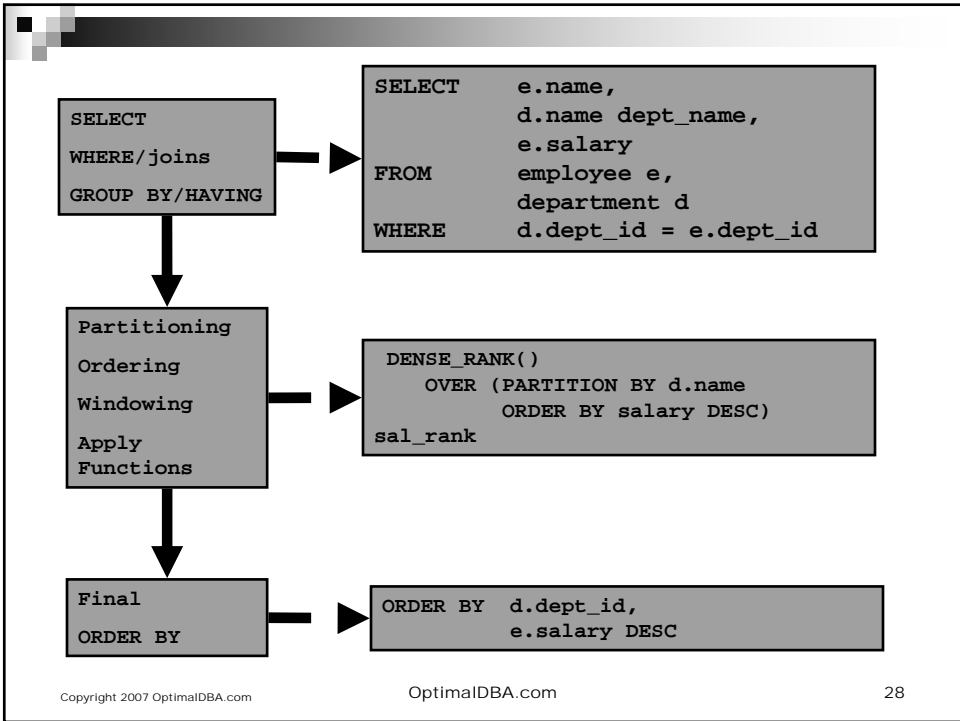


Employee	Department	Salary
Allison Ballinger	Administration	\$90,000
William Dietrich	Administration	\$30,000
Rachel Middleton	Administration	\$850,000
Julie Johnson	Sales and Marketing	\$75,000
Larry Wilton	Sales and Marketing	\$30,000
Shelly Walton	Sales and Marketing	\$30,000
Bobby Harris	Sales and Marketing	\$30,000
Allison Dietrich	Sales and Marketing	\$30,000
Andy Schmidt	Sales and Marketing	\$70,000
Henry Parry	Sales and Marketing	\$30,000
Bev George	Sales and Marketing	\$30,000
Oscar Perry	Sales and Marketing	\$45,000
Stan Marsh	Accounting	\$45,000
John Dennis	Accounting	\$46,000
Erika Deeter	Accounting	\$60,000
Thomas Walton	Logistics and Supply	\$25,000
Tina Walton	Logistics and Supply	\$80,000
Eric Kraus	Logistics and Supply	\$27,000
Doug Harris	Logistics and Supply	\$50,000
Billy Yolto	Logistics and Supply	\$30,000
Vincent Johns	Logistics and Supply	\$50,000

Copyright 2007 OptimalDBA.com OptimalDBA.com 26

Employee	Department	Salary
Rachel Middleton	Administration	\$850,000
Allison Ballinger	Administration	\$90,000
William Dietrich	Administration	\$30,000
Julie Johnson	Sales and Marketing	\$75,000
Andy Schmidt	Sales and Marketing	\$70,000
Oscar Perry	Sales and Marketing	\$45,000
Larry Wilton	Sales and Marketing	\$30,000
Shelly Walton	Sales and Marketing	\$30,000
Bobby Harris	Sales and Marketing	\$30,000
Allison Dietrich	Sales and Marketing	\$30,000
Henry Parry	Sales and Marketing	\$30,000
Bev George	Sales and Marketing	\$30,000
Erika Deeter	Accounting	\$60,000
John Dennis	Accounting	\$46,000
Stan Marsh	Accounting	\$45,000
Tina Walton	Logistics and Supply	\$80,000
Doug Harris	Logistics and Supply	\$50,000
Vincent Johns	Logistics and Supply	\$50,000
Billy Yolto	Logistics and Supply	\$30,000
Eric Kraus	Logistics and Supply	\$27,000
Thomas Walton	Logistics and Supply	\$25,000

Copyright 2007 OptimalDBA.com OptimalDBA.com 27



Employee	Department	Salary	Rank
Rachel Middleton	Administration	\$850,000	1
Allison Ballinger	Administration	\$90,000	2
William Dietrich	Administration	\$30,000	3
Julie Johnson	Sales and Marketing	\$75,000	1
Andy Schmidt	Sales and Marketing	\$70,000	2
Oscar Perry	Sales and Marketing	\$45,000	3
Larry Wilton	Sales and Marketing	\$30,000	4
Shelly Walton	Sales and Marketing	\$30,000	4
Bobby Harris	Sales and Marketing	\$30,000	4
Allison Dietrich	Sales and Marketing	\$30,000	4
Henry Parry	Sales and Marketing	\$30,000	4
Bev George	Sales and Marketing	\$30,000	4
Erika Deeter	Accounting	\$60,000	1
John Dennis	Accounting	\$46,000	2
Stan Marsh	Accounting	\$45,000	3
Tina Walton	Logistics and Supply	\$80,000	1
Doug Harris	Logistics and Supply	\$50,000	2
Vincent Johns	Logistics and Supply	\$50,000	2
Billy Yolto	Logistics and Supply	\$30,000	3
Eric Kraus	Logistics and Supply	\$27,000	4
Thomas Walton	Logistics and Supply	\$25,000	5

Copyright 2007 OptimalDBA.com

OptimalDBA.com

29

Lag/Lead

- Find a value in a row before/after the current row
 - Offset - relative position
 - Default - value if offset not in partition
- Can be used to calculate deltas between values
 - Change in sales from previous month

Copyright 2007 OptimalDBA.com

OptimalDBA.com

30

```

SELECT TO_CHAR(order_month, 'YYYY') sales_year,
       SUM(monthly_sales) yearly_sales
FROM   salesperson_monthly_sales
WHERE  salesperson = 'Shelly Walton'
GROUP BY salesperson, TO_CHAR(order_month, 'YYYY')
ORDER BY sales_year

```

Year	Yearly Sales
1998	\$120,002,727.18
1999	\$204,917,678.91
2000	\$208,571,168.94
2001	\$207,037,098.69
2002	\$212,507,814.70
2003	\$201,995,910.54
2004	\$200,247,848.97
2005	\$209,680,630.38
2006	\$185,797,512.12

Copyright 2007 OptimalDBA.com

OptimalDBA.com

31

Lag

```

SELECT TO_CHAR(order_month, 'YYYY') sales_year,
       SUM(monthly_sales) yearly_sales,
       LAG(SUM(monthly_sales))
         OVER (ORDER BY TO_CHAR(order_month, 'YYYY'))
         prev_yearly_sales
FROM   salesperson_monthly_sales
WHERE  salesperson = 'Shelly Walton'
GROUP BY salesperson, TO_CHAR(order_month, 'YYYY')
ORDER BY sales_year

```

Year	Yearly Sales	Prior Year Sales
1998	\$120,002,727.18	
1999	\$204,917,678.91	\$120,002,727.18
2000	\$208,571,168.94	\$204,917,678.91
2001	\$207,037,098.69	\$208,571,168.94
2002	\$212,507,814.70	\$207,037,098.69
2003	\$201,995,910.54	\$212,507,814.70
2004	\$200,247,848.97	\$201,995,910.54
2005	\$209,680,630.38	\$200,247,848.97
2006	\$185,797,512.12	\$209,680,630.38

Copyright 2007 OptimalDBA.com

OptimalDBA.com

32

Lag

```

SELECT TO_CHAR(order_month, 'YYYY') sales_year,
SUM(monthly_sales) yearly_sales,
LAG(SUM(monthly_sales))
  OVER (ORDER BY TO_CHAR(order_month, 'YYYY'))
      prev_yearly_sales
FROM   salesperson_monthly_sales
WHERE  salesperson = 'Shelly Walton'
GROUP BY salesperson, TO_CHAR(order_month, 'YYYY')
ORDER BY sales_year

```

Year	Yearly Sales	Prior Year Sales
1998	\$120,002,727.18	
1999	\$204,917,678.91	\$120,002,727.18
2000	\$208,571,168.94	\$204,917,678.91
2001	\$207,037,098.69	\$208,571,168.94
2002	\$212,507,814.70	\$207,037,098.69
2003	\$201,995,910.54	\$212,507,814.70
2004	\$200,247,848.97	\$201,995,910.54
2005	\$209,680,630.38	\$200,247,848.97
2006	\$185,797,512.12	\$209,680,630.38

Copyright 2007 OptimalDBA.com

OptimalDBA.com

33

Row Offset

Default Value

```

SELECT TO_CHAR(order_month, 'YYYY') sales_year,
SUM(monthly_sales) yearly_sales,
LAG(SUM(monthly_sales),1,0)
  OVER (ORDER BY TO_CHAR(order_month, 'YYYY'))
      prev_yearly_sales
FROM   salesperson_monthly_sales
WHERE  salesperson = 'Shelly Walton'
GROUP BY salesperson, TO_CHAR(order_month, 'YYYY')
ORDER BY sales_year

```

Year	Yearly Sales	Prior Year Sales
1998	\$120,002,727.18	\$.00
1999	\$204,917,678.91	\$120,002,727.18
2000	\$208,571,168.94	\$204,917,678.91
2001	\$207,037,098.69	\$208,571,168.94
2002	\$212,507,814.70	\$207,037,098.69
2003	\$201,995,910.54	\$212,507,814.70
2004	\$200,247,848.97	\$201,995,910.54
2005	\$209,680,630.38	\$200,247,848.97
2006	\$185,797,512.12	\$209,680,630.38

Copyright 2007 OptimalDBA.com

OptimalDBA.com

34

```

SELECT TO_CHAR(order_month, 'YYYY') sales_year,
SUM(monthly_sales) yearly_sales,
LAG(SUM(monthly_sales),1,0)
  OVER (ORDER BY TO_CHAR(order_month, 'YYYY')) prev_yearly_sales,
(SUM(monthly_sales)) - (LAG(SUM(monthly_sales))
  OVER (ORDER BY TO_CHAR(order_month, 'YYYY')))
  yearly_sales_change
FROM salesperson_monthly_sales
WHERE salesperson = 'Shelly Walton'
GROUP BY salesperson, TO_CHAR(order_month, 'YYYY')
ORDER BY sales_year

```

Year	Yearly Sales	Prior Year Sales	Change from Prior
1998	\$120,002,727.18		\$.00
1999	\$204,917,678.91	\$120,002,727.18	\$84,914,951.73
2000	\$208,571,168.94	\$204,917,678.91	\$3,653,490.03
2001	\$207,037,098.69	\$208,571,168.94	-\$1,534,070.25
2002	\$212,507,814.70	\$207,037,098.69	\$5,470,716.01
2003	\$201,995,910.54	\$212,507,814.70	-\$10,511,904.16
2004	\$200,247,848.97	\$201,995,910.54	-\$1,748,061.57
2005	\$209,680,630.38	\$200,247,848.97	\$9,432,781.41
2006	\$185,797,512.12	\$209,680,630.38	-\$23,883,118.26

Copyright 2007 OptimalDBA.com

OptimalDBA.com

35

```

SELECT TO_CHAR(order_month, 'YYYY') sales_year,
SUM(monthly_sales) yearly_sales,
(((SUM(monthly_sales)) /
  (LAG(SUM(monthly_sales))
  OVER (ORDER BY TO_CHAR(order_month, 'YYYY')))) - 1) * 100
  pct_yearly_sales_change
FROM salesperson_monthly_sales
WHERE salesperson = 'Shelly Walton'
GROUP BY salesperson, TO_CHAR(order_month, 'YYYY')
ORDER BY sales_year
ORDER BY sales_year

```

Year	Yearly Sales	% Change from Prior
1998	\$120,002,727.18	
1999	\$204,917,678.91	70.76
2000	\$208,571,168.94	1.78
2001	\$207,037,098.69	-.74
2002	\$212,507,814.70	2.64
2003	\$201,995,910.54	-4.95
2004	\$200,247,848.97	-.87
2005	\$209,680,630.38	4.71
2006	\$185,797,512.12	-11.39

Copyright 2007 OptimalDBA.com

OptimalDBA.com

36

```

SELECT  salesperson,
        order_month,
        SUM(monthly_sales) AS month_SALES,
        LAG(SUM(monthly_sales),1,0)
          OVER (ORDER BY salesperson, order_month)
          AS prior_month_SALES
FROM    salesperson_monthly_sales
WHERE   order_month BETWEEN '01-JAN-05' AND '31-DEC-05'
GROUP BY salesperson, order_month
ORDER BY salesperson, order_month

```

↓

Allison Dietrich	01-AUG-05	\$16,578,612.72	\$17,967,753.94
Allison Dietrich	01-SEP-05	\$15,888,356.88	\$16,578,612.72
Allison Dietrich	01-OCT-05	\$15,978,969.96	\$15,888,356.88
Allison Dietrich	01-NOV-05	\$15,056,968.50	\$15,978,969.96
Allison Dietrich	01-DEC-05	\$18,743,166.06	\$15,056,968.50
Bev George	01-JAN-05	\$19,998,563.59	\$18,743,166.06
Bev George	01-FEB-05	\$15,399,397.37	\$19,998,563.59
Bev George	01-MAR-05	\$20,042,537.58	\$15,399,397.37
Bev George	01-APR-05	\$18,811,220.29	\$20,042,537.58
Bev George	01-MAY-05	\$17,060,252.02	\$18,811,220.29

Copyright 2007 OptimalDBA.com

OptimalDBA.com

37

```

SELECT  salesperson,
        order_month,
        SUM(monthly_sales) AS month_SALES,
        LAG(SUM(monthly_sales),1,0)
          OVER (ORDER BY salesperson, order_month)
          AS prior_month_SALES
FROM    salesperson_monthly_sales
WHERE   order_month BETWEEN '01-JAN-05' AND '31-DEC-05'
GROUP BY salesperson, order_month
ORDER BY salesperson, order_month

```

↓

Allison Dietrich	01-AUG-05	\$16,578,612.72	\$17,967,753.94
Allison Dietrich	01-SEP-05	\$15,888,356.88	\$16,578,612.72
Allison Dietrich	01-OCT-05	\$15,978,969.96	\$15,888,356.88
Allison Dietrich	01-NOV-05	\$15,056,968.50	\$15,978,969.96
Allison Dietrich	01-DEC-05	\$18,743,166.06	\$15,056,968.50
Bev George	01-JAN-05	\$19,998,563.59	\$18,743,166.06
Bev George	01-FEB-05	\$15,399,397.37	\$19,998,563.59
Bev George	01-MAR-05	\$20,042,537.58	\$15,399,397.37
Bev George	01-APR-05	\$18,811,220.29	\$20,042,537.58
Bev George	01-MAY-05	\$17,060,252.02	\$18,811,220.29

Copyright 2007 OptimalDBA.com

OptimalDBA.com

38

```

SELECT    salesperson,
          order_month,
          SUM(monthly_sales) AS month_SALES,
          LAG(SUM(monthly_sales),1,0)
            OVER (PARTITION BY salesperson
                  ORDER BY order_month)
            AS prior_month_SALES
FROM      salesperson_monthly_sales
WHERE     order_month BETWEEN '01-JAN-05' AND '31-DEC-05'
GROUP BY salesperson, order_month
ORDER BY salesperson, order_month

```

↓

Allison Dietrich	01-AUG-05	\$16,578,612.72	\$17,967,753.94
Allison Dietrich	01-SEP-05	\$15,888,356.88	\$16,578,612.72
Allison Dietrich	01-OCT-05	\$15,978,969.96	\$15,888,356.88
Allison Dietrich	01-NOV-05	\$15,056,968.50	\$15,978,969.96
Allison Dietrich	01-DEC-05	\$18,743,166.06	\$15,056,968.50
Bev George	01-JAN-05	\$19,998,563.59	\$.00
Bev George	01-FEB-05	\$15,399,397.37	\$19,998,563.59
Bev George	01-MAR-05	\$20,042,537.58	\$15,399,397.37
Bev George	01-APR-05	\$18,811,220.29	\$20,042,537.58
Bev George	01-MAY-05	\$17,060,252.02	\$18,811,220.29

Using Analytical SQL and Statspack Data

- Statistic values for a given period
- Compare statistic values for 2 periods
- Top SQL

Simple Delta Calculation

- LAG function
 - Returns the specified value from a previous row
 - Subtract from current row value to calculate delta
- NULL delta values should be ignored

Timed Event Delta

- Example is from recent test of san configurations
- Using the ucomment field to deliniate tests
- Timed Events
 - db file sequential read (single block)
 - db file scattered read (multi block)

```

SELECT      sl.ucomment,
            wl.event,
            sl.snap_id,
            wl.total_waits,
            wl.time_waited_micro
FROM        stats$snapshot sl,
            stats$system_event wl
WHERE       sl.snap_id BETWEEN 313 AND 320
            AND sl.snap_id = wl.snap_id
            AND wl.event = 'db file sequential read'
ORDER BY   wl.event, sl.snap_id

```

UCOMMENT	EVENT	SNAP_ID	TOTAL_WAITS	TIME_WAITED_MICRO
PRODDB1 test	db file sequential read	313	1344	5521259
PRODDB1 test	db file sequential read	314	4717682	55862171
PRODDB1 test	db file sequential read	315	17958436	212135571
PRODDB1 test	db file sequential read	316	32697239	390437955
PRODDB1 test	db file sequential read	317	47420163	571023726
PRODDB1 test	db file sequential read	318	59789414	752841065
PRODDB1 test	db file sequential read	319	74283132	943025186
PRODDB1 test	db file sequential read	320	88428381	1119908896

```

SELECT      s1.ucomment,
            w1.event,
            s1.snap_id,
            w1.total_waits,
            LAG(w1.total_waits)
              OVER (ORDER BY s1.snap_id) prev_val,
            LEAD(w1.total_waits)
              OVER (ORDER BY s1.snap_id) next_val
FROM        stats$snapshot s1,
            stats$system_event w1
WHERE       s1.snap_id BETWEEN 313 AND 339
AND         s1.snap_id = w1.snap_id
AND         w1.event = 'db file sequential read'
ORDER BY   w1.event, s1.snap_id

```

UCOMMENT	EVENT	SNAP	TOTAL_WAITS	PREV_VAL	NEXT_VAL
PRODDB1 test	db file sequential read	313	1344		4717682
PRODDB1 test	db file sequential read	314	4717682	1344	17958436
PRODDB1 test	db file sequential read	315	17958436	4717682	32697239
PRODDB1 test	db file sequential read	316	32697239	17958436	47420163
PRODDB1 test	db file sequential read	317	47420163	32697239	59789414
PRODDB1 test	db file sequential read	318	59789414	47420163	74283132
PRODDB1 test	db file sequential read	319	74283132	59789414	88428381
PRODDB1 test	db file sequential read	320	88428381	74283132	103814525

```

SELECT      s1.ucomment,
            w1.event,
            s1.snap_id,
            w1.total_waits,
            LAG(w1.total_waits)
              OVER (ORDER BY s1.snap_id) prev_val,
            w1.total_waits -
            LAG(w1.total_waits)
              OVER (ORDER BY s1.snap_id) delta_val
FROM        stats$snapshot s1,
            stats$system_event w1
WHERE       s1.snap_id BETWEEN 313 AND 320
            AND s1.snap_id = w1.snap_id
            AND w1.event = 'db file sequential read'
ORDER BY   w1.event, s1.snap_id

```

UCOMMENT	EVENT	SNAP	TOTAL_WAITS	PREV_VAL	DELTA_VAL
PRODDb1 test	db file sequential read	313	1344		
PRODDb1 test	db file sequential read	314	4717682	1344	4716338
PRODDb1 test	db file sequential read	315	17958436	4717682	13240754
PRODDb1 test	db file sequential read	316	32697239	17958436	14738803
PRODDb1 test	db file sequential read	317	47420163	32697239	14722924
PRODDb1 test	db file sequential read	318	59789414	47420163	12369251
PRODDb1 test	db file sequential read	319	74283132	59789414	14493718
PRODDb1 test	db file sequential read	320	88428381	74283132	14145249

```

WITH comment_snap_range AS
( SELECT      ucomment, MIN(snap_id) snapid
  FROM        stats$snapshot
 WHERE       ucomment IS NOT NULL
 GROUP BY    ucomment
 UNION
 SELECT      ucomment, MAX(snap_id) snapid
  FROM        stats$snapshot
 WHERE       ucomment IS NOT NULL
 GROUP BY    ucomment ),
test_event_time AS
( SELECT      c1.ucomment, w1.event,
              c1.snapid, w1.time_waited_micro
  FROM        comment_snap_range c1, stats$system_event w1
 WHERE       c1.snapid = w1.snapid
            AND w1.event in ('db file sequential read',
                              'db file scattered read')
 ORDER BY    (SELECT MIN(s2.snap_id)
              FROM stats$snapshot s2
              WHERE s2.ucomment = c1.ucomment),
              w1.event, c1.snapid )

```

```

SELECT ucomment,
       event,
       snapid,
       time_waited_micro,
       LAG(time_waited_micro)
         OVER (ORDER BY rownum) prev_twm
FROM   test_event_time

```

Test	Event	Snap	Time (micro)	Prev (micro)
PRODDb1 test	db file scattered read	313	401714	
PRODDb1 test	db file scattered read	339	17289525	401714
PRODDb1 test	db file sequential read	313	5521259	17289525
PRODDb1 test	db file sequential read	339	4428489413	5521259
R5 Test	db file scattered read	540	80773789	4428489413
R5 Test	db file scattered read	566	97958383	80773789
R5 Test	db file sequential read	540	4523917087	97958383
R5 Test	db file sequential read	566	8408009276	4523917087
R10 Test	db file scattered read	586	172789039	8408009276
R10 Test	db file scattered read	611	182687280	172789039
R10 Test	db file sequential read	586	9442385853	182687280
R10 Test	db file sequential read	611	13740256353	9442385853

```

SELECT ucomment,
       event,
       snapid,
       time_waited_micro,
       LAG(time_waited_micro)
         OVER (PARTITION BY ucomment, event
              ORDER BY rownum) prev_twm,
       time_waited_micro -
       LAG(time_waited_micro)
         OVER (PARTITION BY ucomment, event
              ORDER BY rownum) delta_twm
FROM   test_event_time

```

UCCOMMENT	EVENT	Snap	TIME_WAITED_MICRO	PREV_TWM	DELTA_TWM
PRODDb1 test	db file scattered read	313	401714		
PRODDb1 test	db file scattered read	339	17289525	401714	16887811
PRODDb1 test	db file sequential read	313	5521259		
PRODDb1 test	db file sequential read	339	4428489413	5521259	4422968154
R5 Test	db file scattered read	540	80773789		
R5 Test	db file scattered read	566	97958383	80773789	17184594
R5 Test	db file sequential read	540	4523917087		
R5 Test	db file sequential read	566	8408009276	4523917087	3884092189
R10 Test	db file scattered read	586	172789039		
R10 Test	db file scattered read	611	182687280	172789039	9898241
R10 Test	db file sequential read	586	9442385853		
R10 Test	db file sequential read	611	13740256353	9442385853	4297870500

Test	Event	Delta (micro)
PRODDb1 test	db file scattered read	16887811
PRODDb1 test	db file sequential read	4422968154
R5 Test	db file scattered read	17184594
R5 Test	db file sequential read	3884092189
R10 Test	db file scattered read	9898241
R10 Test	db file sequential read	4297870500

Comparing Between 2 Periods

- Using LAG and Partitioning, we can determine accurate deltas
- To compare 2 periods, we use the same logic

```
SELECT sy.snap_id,  
       sy.statistic# statistic#,  
       sy.name statname,  
       sy.value - (LAG(sy.value)  
                   OVER (PARTITION BY sy.name  
                         ORDER BY sy.snap_id)) statdelta  
FROM   stats$sysstat sy  
WHERE  sy.snap_id IN (12208,12599,13480,13843)  
AND    sy.name IN  
       ('consistent gets','consistent changes',  
        'db block gets', 'db block changes')  
ORDER BY sy.name, sy.snap_id
```

SNAP_ID	STATISTIC#	STATNAME	STATDELTA
12208	53	consistent changes	
12599	53	consistent changes	456,546,852
13480	53	consistent changes	16,110,399,791
13843	53	consistent changes	4,188,844,228
12208	44	consistent gets	
12599	44	consistent gets	52,156,187,345
13480	44	consistent gets	173,994,072,540
13843	44	consistent gets	126,266,906,068
12208	52	db block changes	
12599	52	db block changes	429,129,811
13480	52	db block changes	1,985,694,724
13843	52	db block changes	2,341,341,202
12208	41	db block gets	
12599	41	db block gets	506,952,728
13480	41	db block gets	2,195,528,867
13843	41	db block gets	2,988,539,957

```

SELECT CASE
    WHEN sy.snap_id IN (12208,12599) THEN 1
    WHEN sy.snap_id IN (13480,13843) THEN 2
    ELSE 0
END snap_range,
sy.snap_id,
sy.statistic# statistic#,
sy.name statname,
sy.value
FROM stats$sysstat sy
WHERE sy.snap_id IN (12208,12599,13480,13843)
AND sy.name IN ('consistent gets','consistent
changes',
'db block gets', 'db block
changes')
ORDER BY statname, snap_id

```

RANGE	SNAP_ID	STAT#	STATNAME	VALUE
1	12208	53	consistent changes	18,994,651,171
1	12599	53	consistent changes	19,451,198,023
2	13480	53	consistent changes	35,561,597,814
2	13843	53	consistent changes	39,750,442,042
1	12208	44	consistent gets	297,560,967,361
1	12599	44	consistent gets	349,717,154,706
2	13480	44	consistent gets	523,711,227,246
2	13843	44	consistent gets	649,978,133,314
1	12208	52	db block changes	3,896,282,280
1	12599	52	db block changes	4,325,412,091
2	13480	52	db block changes	6,311,106,815
2	13843	52	db block changes	8,652,448,017
1	12208	41	db block gets	4,905,015,633
1	12599	41	db block gets	5,411,968,361
2	13480	41	db block gets	7,607,497,228
2	13843	41	db block gets	10,596,037,185

```

SELECT sy.snap_id,
       sy.statistic# statistic#,
       sy.name statname,
       sy.value - (LAG(sy.value)
                  OVER (PARTITION BY sy.name
                        ORDER BY sy.snap_id)) statdelta
FROM   stats$sysstat sy
WHERE  sy.snap_id IN (12208,12599,13480,13843)
AND    sy.name IN
       ('consistent gets','consistent changes',
       'db block gets', 'db block changes')
ORDER BY sy.name, sy.snap_id

```

SNAP_ID	STATISTIC#	STATNAME	STATDELTA
12208	53	consistent changes	
12599	53	consistent changes	456,546,852
13480	53	consistent changes	16,110,399,791
13843	53	consistent changes	4,188,844,228
12208	44	consistent gets	
12599	44	consistent gets	52,156,187,345
13480	44	consistent gets	173,994,072,540
13843	44	consistent gets	126,266,906,068
12208	52	db block changes	
12599	52	db block changes	429,129,811
13480	52	db block changes	1,985,694,724
13843	52	db block changes	2,341,341,202
12208	41	db block gets	
12599	41	db block gets	506,952,728
13480	41	db block gets	2,195,528,867
13843	41	db block gets	2,988,539,957

Copyright 2007 OptimalDBA.com

OptimalDBA.com

61

```

SELECT snap_range range,
       snap_id,
       statistic# stat#,
       statname,
       value - (LAG(value) OVER
                (PARTITION BY statname,snap_range
                 ORDER BY snap_id)) statdelta
FROM   ( SELECT CASE
           WHEN sy.snap_id IN (12208,12599) THEN 1
           WHEN sy.snap_id IN (13480,13843) THEN 2
           ELSE 0
         END snap_range,
         sy.snap_id,
         sy.statistic# statistic#,
         sy.name statname,
         sy.value
       FROM   stats$sysstat sy
       WHERE  sy.snap_id IN (12208,12599,13480,13843)
         AND  sy.name IN ('consistent gets',
                          'consistent changes',
                          'db block gets', 'db block changes')
     )
ORDER BY statname, snap_id

```

Copyright 2007 OptimalDBA.com

OptimalDBA.com

62

RANGE	SNAP_ID	STAT#	STATNAME	STATDELTA
1	12208	53	consistent changes	
1	12599	53	consistent changes	456,546,852
2	13480	53	consistent changes	
2	13843	53	consistent changes	4,188,844,228
1	12208	44	consistent gets	
1	12599	44	consistent gets	52,156,187,345
2	13480	44	consistent gets	
2	13843	44	consistent gets	126,266,906,068
1	12208	52	db block changes	
1	12599	52	db block changes	429,129,811
2	13480	52	db block changes	
2	13843	52	db block changes	2,341,341,202
1	12208	41	db block gets	
1	12599	41	db block gets	506,952,728
2	13480	41	db block gets	
2	13843	41	db block gets	2,988,539,957

```

SELECT range,
       snap_id,
       stat#,
       statname,
       statdelta,
       (LAG(statdelta) OVER
        (PARTITION BY statname
         ORDER BY snap_id)) prevstatdelta
FROM   snap_range_lio
WHERE  statdelta IS NOT NULL
ORDER BY statname, snap_id

```

RANGE	SNAP_ID	STAT#	STATNAME	STATDELTA	PREVSTATDELTA
1	12599	53	consistent changes	456,546,852	
2	13843	53	consistent changes	4,188,844,228	456,546,852
1	12599	44	consistent gets	52,156,187,345	
2	13843	44	consistent gets	126,266,906,068	52,156,187,345
1	12599	52	db block changes	429,129,811	
2	13843	52	db block changes	2,341,341,202	429,129,811
1	12599	41	db block gets	506,952,728	
2	13843	41	db block gets	2,988,539,957	506,952,728

```

SELECT snap_range range,
       snap_id,
       statistic# stat#,
       statname,
       value - (LAG(value) OVER
                (PARTITION BY statname,snap_range
                 ORDER BY snap_id)) statdelta
FROM   ( SELECT CASE
           WHEN sy.snap_id IN (12208,12599) THEN 1
           WHEN sy.snap_id IN (13480,13843) THEN 2
           ELSE 0
         END snap_range,
         sy.snap_id,
         sy.statistic# statistic#,
         sy.name statname,
         sy.value
       FROM   stats$sysstat sy
       WHERE  sy.snap_id IN (12208,12599,13480,13843)
         AND  sy.name IN ('consistent gets',
                        'consistent changes',
                        'db block gets', 'db block changes')
     )
ORDER BY statname, snap_id

```

RANGE	SNAP_ID	STAT#	STATNAME	STATDELTA
1	12208	53	consistent changes	
1	12599	53	consistent changes	456,546,852
2	13480	53	consistent changes	
2	13843	53	consistent changes	4,188,844,228
1	12208	44	consistent gets	
1	12599	44	consistent gets	52,156,187,345
2	13480	44	consistent gets	
2	13843	44	consistent gets	126,266,906,068
1	12208	52	db block changes	
1	12599	52	db block changes	429,129,811
2	13480	52	db block changes	
2	13843	52	db block changes	2,341,341,202
1	12208	41	db block gets	
1	12599	41	db block gets	506,952,728
2	13480	41	db block gets	
2	13843	41	db block gets	2,988,539,957

```

WITH snap_range_lio AS
( SELECT snap_range range,
  snap_id,
  statistic# stat#,
  statname,
  value - (LAG(value) OVER
    (PARTITION BY statname,snap_range
    ORDER BY snap_id)) statdelta
FROM ( SELECT CASE
  WHEN sy.snap_id IN (12208,12599) THEN 1
  WHEN sy.snap_id IN (13480,13843) THEN 2
  ELSE 0
  END snap_range,
  sy.snap_id,
  sy.statistic# statistic#,
  sy.name statname,
  sy.value
FROM stats$sysstat sy
WHERE sy.snap_id IN (12208,12599,13480,13843)
AND sy.name IN ('consistent gets',
  'consistent changes',
  'db block gets',
  'db block changes')
)

```

```

snap_lio_summary AS
( SELECT range,
      snap_id,
      stat#,
      statname,
      (LAG(statdelta) OVER
        (PARTITION BY statname
         ORDER BY snap_id)) statdelta1,
      statdelta statdelta2,
      statdelta -
      (LAG(statdelta) OVER
        (PARTITION BY statname
         ORDER BY snap_id))
      rangestatdelta
FROM   snap_range_lio
WHERE  statdelta IS NOT NULL
)

```

```

SELECT statname,
      statdelta1,
      statdelta2,
      rangestatdelta,
      (rangestatdelta / statdelta1) * 100 rangestatpct
FROM   snap_lio_summary
WHERE  rangestatdelta is not null
ORDER BY statname, snap_id

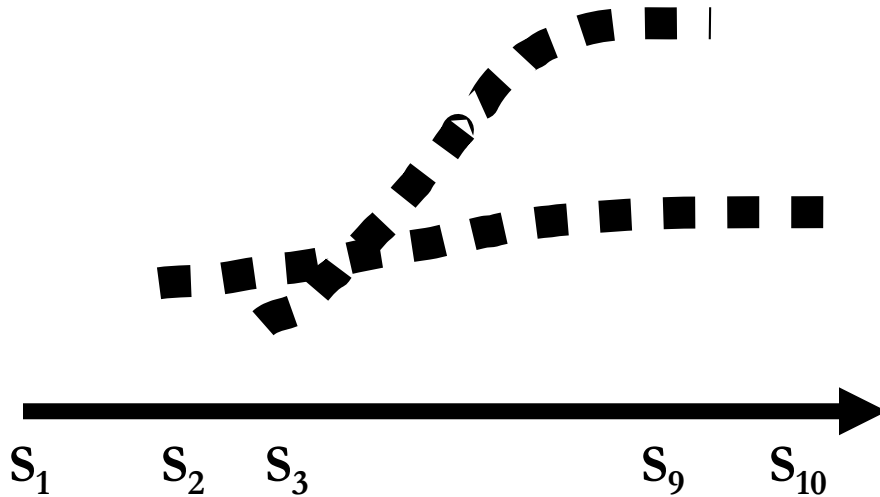
```

Statistic	Range 1	Range 2	Range Delta	Delta %
consistent changes	456,546,852	4,188,844,228	3,732,297,376	817.51
consistent gets	52,156,187,345	126,266,906,068	74,110,718,723	142.09
db block changes	429,129,811	2,341,341,202	1,912,211,391	445.60
db block gets	506,952,728	2,988,539,957	2,481,587,229	489.51

SQL Performance

- What are the bad statements?
- If a statement was not captured at both snapshots, it is not reported
 - Even if it was the most resource consuming statement of the period
 - Why would it not be captured?
 - Thresholds
 - Not in shared pool
- 10gR1 and older - stats reported only when statements completes!

SQL Reporting



Copyright 2007 OptimalDBA.com

OptimalDBA.com

73

SQL Performance

- Capturing an "accurate" picture requires the dreaded cartesian product
 - A list of all snapshot ids with every hash value
- Gather deltas for sql statistics for each id and hash value combination (0 if null)
- SUM values of deltas
- Report only Top N

Copyright 2007 OptimalDBA.com

OptimalDBA.com

74

SQL Hash Value	Executions	Rows	Elapsed Seconds	ELA
1734005396	1,958	1,958	596,336	1
3105251070	1,855	8,119	515,942	2
18145424	5,137,178	5,137,173	393,015	3
1864143593	5,203,351	5,203,350	345,444	4
1591652717	1,163,583	1,163,583	219,862	5
2276562963	11,188	11,187	182,776	6
2263873416	9,945	9,945	178,371	7
4212608483	1,969	0	132,684	8
3730474290	671	22,614,602	122,956	9
3312786039	603	603	122,441	10

Next Steps

- Write your own spreport
- Use statspack data to populate dss structures

- Questions?
 - daniel.fink@optimaldba.com
- Presentations/Code
 - www.optimaldba.com
- RMOUG Training Days 2008
 - February 12 - 14, 2008
 - Denver, Colorado
 - www.rmoug.org