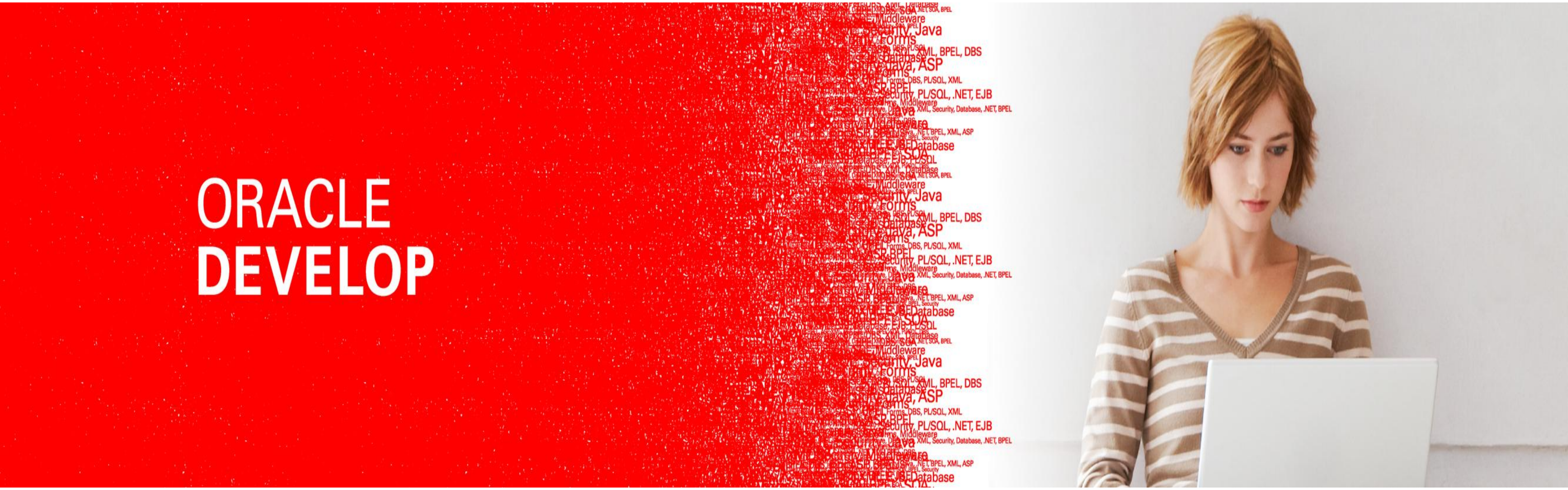
ORACLE®



“Mirror, Mirror: Tell me Why my Application Sucks”

Kuassi Mensah
Database Access Services, Database APIs, and Net Services

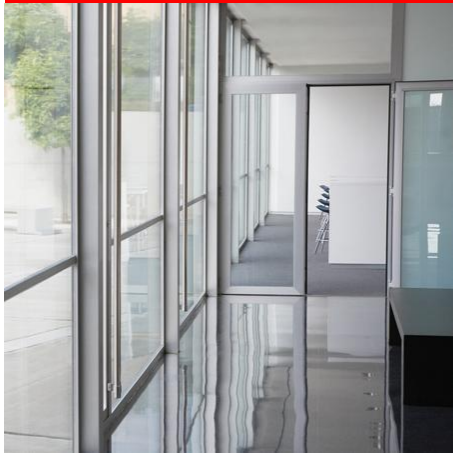
The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions.

The development, release, and timing of any features or functionality described for Oracle’s products remains at the sole discretion of Oracle.

Focus of this presentation

- *Not* about SQL tuning
- *Not* about Oracle Database instance tuning
- It *is* about using Database performance tools to uncover inefficient database access
- It *is* about implementing best practices for writing applications for efficient Database access
- It *is* about any programming language

Agenda

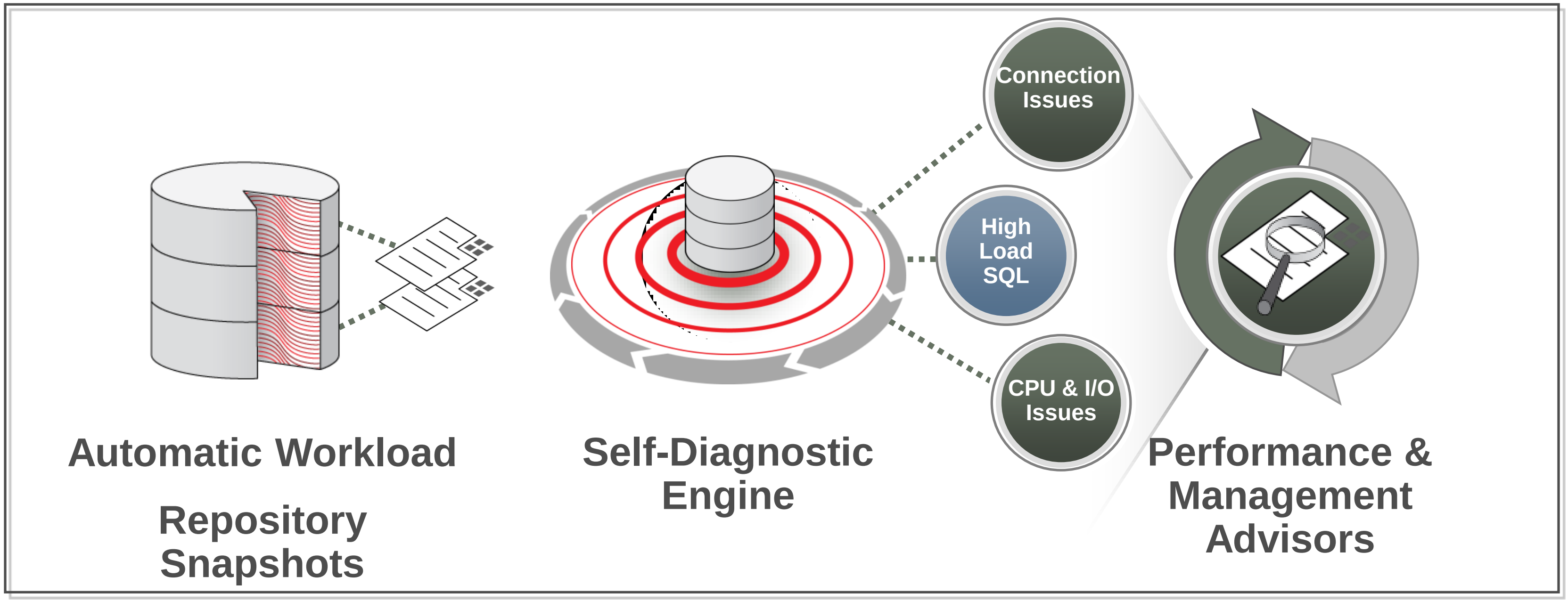


- Database Performance Monitoring Tools
- Use Cases & Best Practices
  - Connections
  - Hard Parses
  - Soft Parses
  - Wrong Default
  - Array DML
  - Stored Procedures
  - Client-side Result Set Caching
  - LOBs

Database Performance Monitoring Tools

AWR and ADDM

Enterprise Manager - Automatic Performance Diagnostics



Automatic Workload

Repository Snapshots

Self-Diagnostic Engine

Performance & Management Advisors

Getting ADDM/AWR Reports

- Create an AWR Snapshot

```
BEGIN
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ();
END;
```

- Run your workload
- Create a second AWR Snapshot

```
BEGIN
DBMS_WORKLOAD_REPOSITORY.CREATE_SNAPSHOT ();
END;
```

- Generate reports

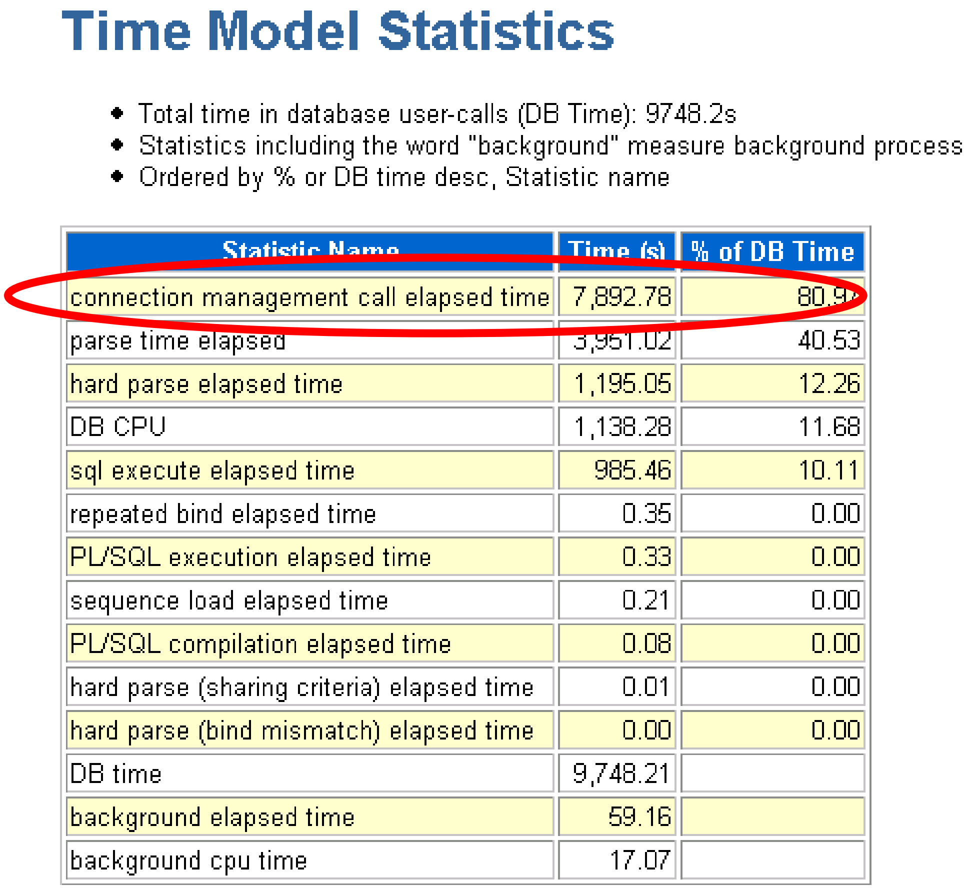
```
@$ORACLE_HOME/rdbms/admin/addmrpt.sql
@$ORACLE_HOME/rdbms/admin/awrrpt.sql
```

Connection Performance

WTF with Connections?

- Top Two out of *“Top Ten Mistakes Found In Oracle Systems”*:
  - Bad Connection Management
  - Bad Use of Cursors and the Shared Pool
- Database Connections expensive to create
  - Spawn O/S process, network connection, several roundtrips
  - Associated database authentication and session creation
- Database Connections are expensive to tear down!
- Repeatedly Connecting/Disconnecting can be a huge scaling issue

Connections Statistics in AWR report



Time Model Statistics

- Total time in database user-calls (DB Time): 9748.2s
- Statistics including the word "background" measure background process
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
connection management call elapsed time	7,892.78	80.97
parse time elapsed	3,951.02	40.53
hard parse elapsed time	1,195.05	12.26
DB CPU	1,138.28	11.68
sql execute elapsed time	985.46	10.11
repeated bind elapsed time	0.35	0.00
PL/SQL execution elapsed time	0.33	0.00
sequence load elapsed time	0.21	0.00
PL/SQL compilation elapsed time	0.08	0.00
hard parse (sharing criteria) elapsed time	0.01	0.00
hard parse (bind mismatch) elapsed time	0.00	0.00
DB time	9,748.21	
background elapsed time	59.16	
background cpu time	17.07	

Connections
ADDM Recommendations

```
Finding 3: Session Connect and Disconnect
Impact is 9.59 active sessions, 80.97% of total activity.
---------------------------------------------------------
Session connect and disconnect calls were consuming significant database time.

   Recommendation 1: Application Analysis
   Estimated benefit is 9.59 active sessions, 80.97% of total activity.
   --------------------------------------------------------------------
   Action
      Investigate application logic for possible reduction of connect and
      disconnect calls. For example, you might use a connection pool scheme in
      the middle tier.
```

Java Universal Connection Pool

```
Main Thread:
        // Create a data source
      PoolDataSource pds = new PoolDataSourceImpl();

      System.out.println ("Connecting to " + url);
      // Set DataSource properties
      pds.setConnectionFactoryClassName("oracle.jdbc.pool.OracleDataSource");
      pds.setURL(url);
      pds.setUser(user);
      pds.setPassword(password);

      pds.setConnectionPoolName(“MyPool");
      pds.setMinPoolSize(10);
      pds.setMaxPoolSize(100); // Set DataSource properties

Thread:
           // Obtain a connection
          connection = dataSource.getConnection();
          // run the workload
          doWork(connection);

          // close the connection when done
          connection.close();
```

Database Resident Connection Pool (DRCP)

C, C++, PHP, Python, Perl

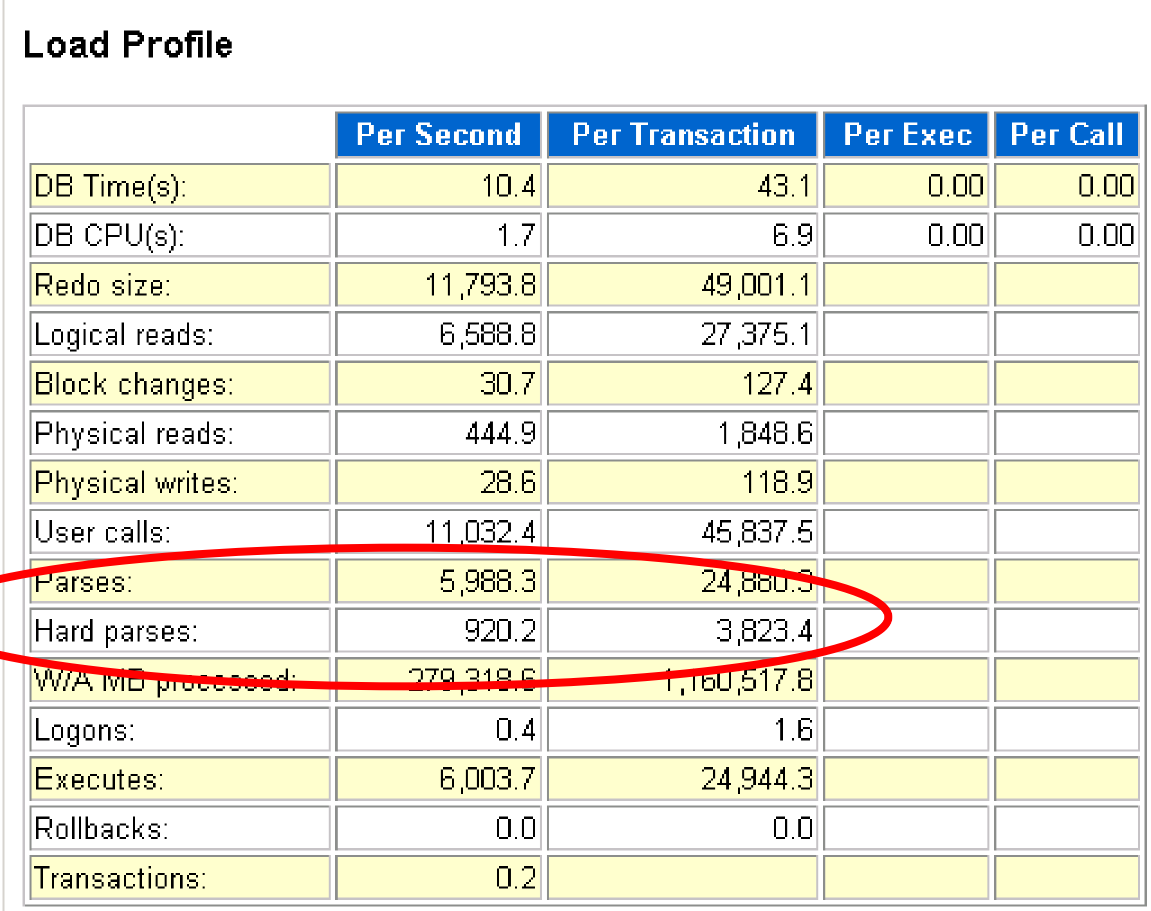
- Scales to tens of thousands of database connections even on a commodity box
- Indispensable for sharing connections across middle tier hosts
- Fallback when there is no application tier connection pooling
- Enable with `dbms_connection_pool.start_pool`
- Connect String
  - `Easy Connect: //localhost:1521/oowlab:POOLED`
  - `TNS Connect String: (SERVER=POOLED)`

Hard Parsing

Hard Parsing

- Hard Parse is expensive
  - Creates shared cursor in SGA
  - Causes library cache latch contention
  - Causes shared pool contention
  - Causes scalability issues

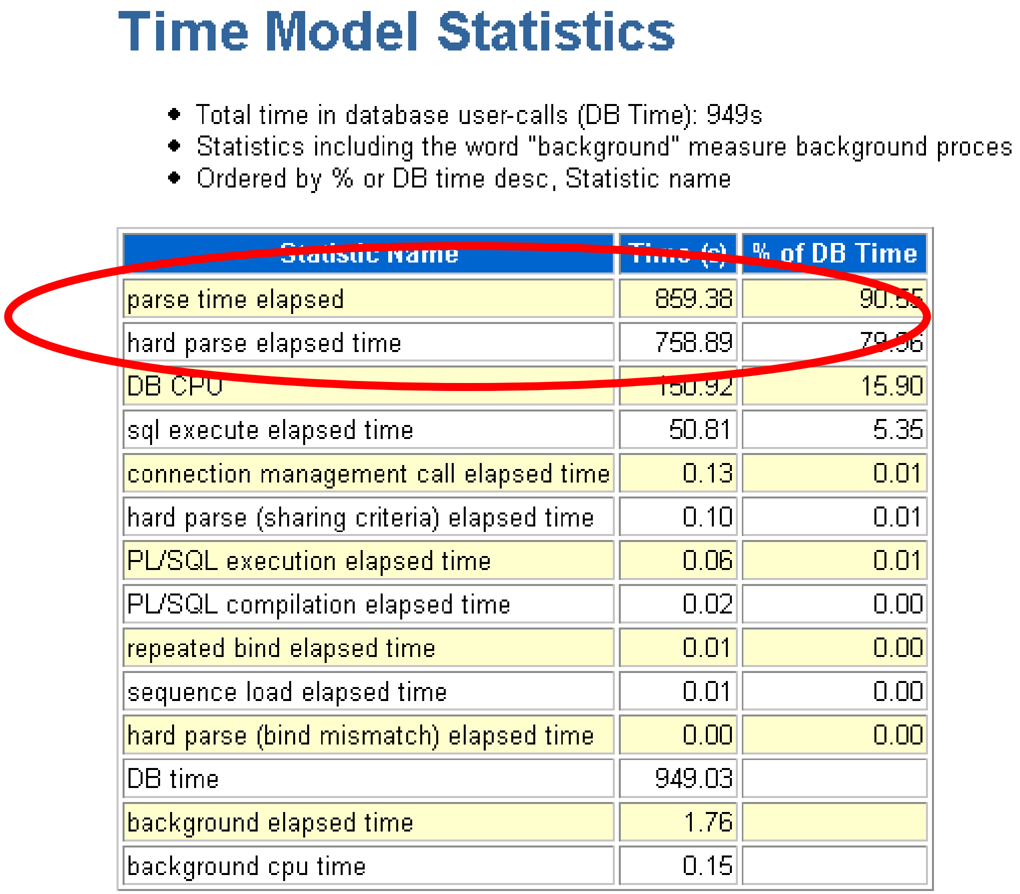
Hard Parsing: AWR report



Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	10.4	43.1	0.00	0.00
DB CPU(s):	1.7	6.9	0.00	0.00
Redo size:	11,793.8	49,001.1		
Logical reads:	6,588.8	27,375.1		
Block changes:	30.7	127.4		
Physical reads:	444.9	1,848.6		
Physical writes:	28.6	118.9		
User calls:	11,032.4	45,837.5		
Parses:	5,988.3	24,880.3		
Hard parses:	920.2	3,823.4		
W/A MB processed:	279,318.6	1,160,517.8		
Logons:	0.4	1.6		
Executes:	6,003.7	24,944.3		
Rollbacks:	0.0	0.0		
Transactions:	0.2			

Hard Parsing: more from the same AWR report



Time Model Statistics

- Total time in database user-calls (DB Time): 949s
- Statistics including the word "background" measure background proces
- Ordered by % or DB time desc, Statistic name

Statistic Name	Time (s)	% of DB Time
parse time elapsed	859.38	90.55
hard parse elapsed time	758.89	79.96
DB CPU	150.92	15.90
sql execute elapsed time	50.81	5.35
connection management call elapsed time	0.13	0.01
hard parse (sharing criteria) elapsed time	0.10	0.01
PL/SQL execution elapsed time	0.06	0.01
PL/SQL compilation elapsed time	0.02	0.00
repeated bind elapsed time	0.01	0.00
sequence load elapsed time	0.01	0.00
hard parse (bind mismatch) elapsed time	0.00	0.00
DB time	949.03	
background elapsed time	1.76	
background cpu time	0.15	

Hard Parsing: ADDM Recommendations

```
Finding 2: Hard Parse Due to Literal Usage
Impact is 8.32 active sessions, 79.74% of total activity.
---------------------------------------------------------
SQL statements were not shared due to the usage of literals. This resulted in
additional hard parses which were consuming significant database time.

   Recommendation 1: Application Analysis
   Estimated benefit is 8.32 active sessions, 79.74% of total activity.
   --------------------------------------------------------------------
   Action
      Investigate application logic for possible use of bind variables instead
      of literals.
   Action
      Alternatively, you may set the parameter "cursor_sharing" to "force".
   Rationale
      At least 39 SQL statements with FORCE_MATCHING_SIGNATURE
      5551823750033335619 and PLAN_HASH_VALUE 1833546154 were found to be
      using literals. Look in V$SQL for examples of such SQL statements.
```

Hard Parsing Best Practices

- Avoid Hard Parsing with Bind Variables
  - Reduces hard parses on the server
  - Reduces risk of SQL Injection: potential security issue

Hard Parsing Best Practices
Bind Variables in Java

- Instead of:

```
String query = "SELECT EMPLOYEE_ID, LAST_NAME, SALARY FROM "
              +"EMPLOYEES WHERE EMPLOYEE_ID = "
              + generateNumber(MIN_EMPLOYEE_ID, MAX_EMPLOYEE_ID);
pstmt = connection.prepareStatement(query);
rs = pstmt.executeQuery();
```

- Change to:

```
String query = "SELECT EMPLOYEE_ID, LAST_NAME, SALARY FROM "
              +"EMPLOYEES WHERE EMPLOYEE_ID = ?";

pstmt = connection.prepareStatement(query);
pstmt.setInt(1, n);
rs = pstmt.executeQuery();
```

Hard Parsing Best Practices

Bind Variables in C (OCI)

```
static char *MY_SELECT = "select employee_id, last_name, salary from \
                                employees where employee_id = :EMPNO";

OCIBind *bndp1;
OCIStmt *stmthp;
ub4 emp_id;

OCIStmtPrepare2 (svchp, &stmthp,          /* returned stmt handle */
                 errhp,                           /* error handle */
                 (const OraText *) MY_SELECT,
                 strlen((char *) MY_SELECT),
                 NULL, 0,            /* tagging parameters:optional */
                 OCI_NTV_SYNTAX, OCI_DEFAULT);
/* bind input parameters */
OCIBindByName(stmthp, &bndp1, errhp, (text *) ":EMPNO",
              -1, &(emp_id), sizeof(emp_id), SQLT_INT,
               NULL, NULL, NULL, 0, NULL, OCI_DEFAULT);
```

Hard Parsing Best Practices
Literal Replacement

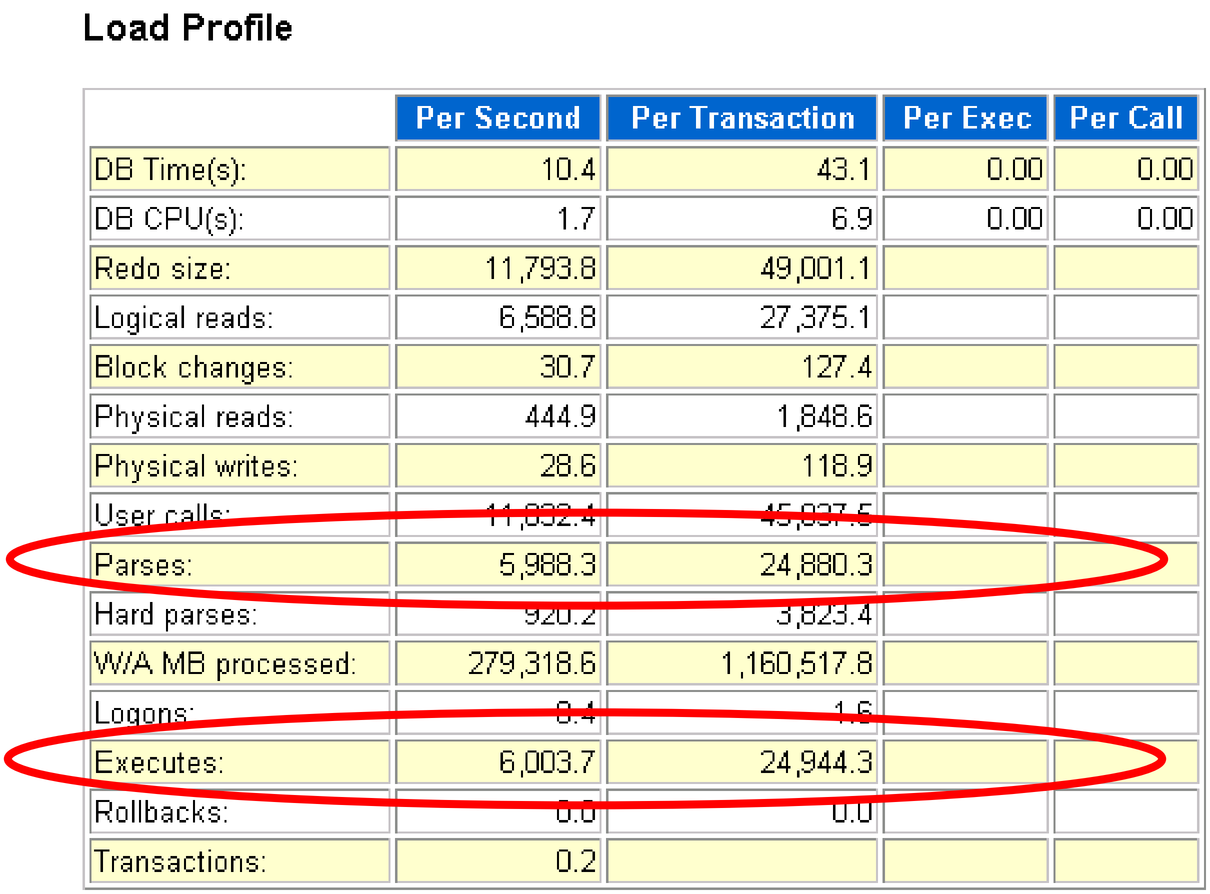
- Fallback if application cannot be changed to use binds
- init.ora parameter
  - `CURSOR_SHARING={FORCE|SIMILAR|EXACT}`
  - Default is `EXACT`

Soft Parsing

Soft Parsing

- Soft Parsing
  - Session executes a statement that exists in shared pool
  - Creates session specific cursor context
  - Repeats metadata processing

Soft Parsing: AWR report



Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	10.4	43.1	0.00	0.00
DB CPU(s):	1.7	6.9	0.00	0.00
Redo size:	11,793.8	49,001.1		
Logical reads:	6,588.8	27,375.1		
Block changes:	30.7	127.4		
Physical reads:	444.9	1,848.6		
Physical writes:	28.6	118.9		
User calls:	11,032.4	45,837.5		
Parses:	5,988.3	24,880.3		
Hard parses:	920.2	3,823.4		
W/A MB processed:	279,318.6	1,160,517.8		
Logons:	0.4	1.6		
Executes:	6,003.7	24,944.3		
Rollbacks:	0.0	0.0		
Transactions:	0.2			

Soft Parsing: ADDM

```
Finding 3: Soft Parse
Impact is 1.1 active sessions, 10.59% of total activity.
--------------------------------------------------------
Soft parsing of SQL statements was consuming significant database time.

   Recommendation 1: Application Analysis
   Estimated benefit is 1.1 active sessions, 10.59% of total activity.
   -------------------------------------------------------------------
   Action
      Investigate application logic to keep open the frequently used cursors.
      Note that cursors are closed by both cursor close calls and session
      disconnects.
```

Soft Parsing Best Practices

- Use Statement Caching
  - Keeps frequently used session cursors open
  - Reduces soft parses on the Server
    - Not only faster but more scalable
  - Cuts repeated metadata processing
  - Consumes less network bandwidth
  - Cuts code path in driver/application tier

Soft Parsing Best Practices

Statement Caching in Java

```
// Obtain a connection
connection = dataSource.getConnection();

// Enable statement caching
((OracleConnection)connection).setStatementCacheSize(20);
((OracleConnection)connection).setImplicitCachingEnabled(true);
```

Soft Parsing Best Practices

Statement Caching in C (OCI)

- Initialize the OCI Session Pool with statement cache

```
ub4       stmt_cachesize = 20;
/* set the statement cache size for all sessions in the pool */
OCIAttrSet(spoolhp, OCI_HTYPE_SPOOL, &stmt_cachesize, 0,
           OCI_ATTR_SPOOL_STMTCACHESIZE, errhp);

/* create a homogeneous session pool */
OCISessionPoolCreate(envhp, errhp,
                     spoolhp,             /* session pool handle */
                     . . .,
                     OCI_SPC_HOMOGENEOUS|
                     OCI_SPC_STMTCACHE);                /* modes */
```

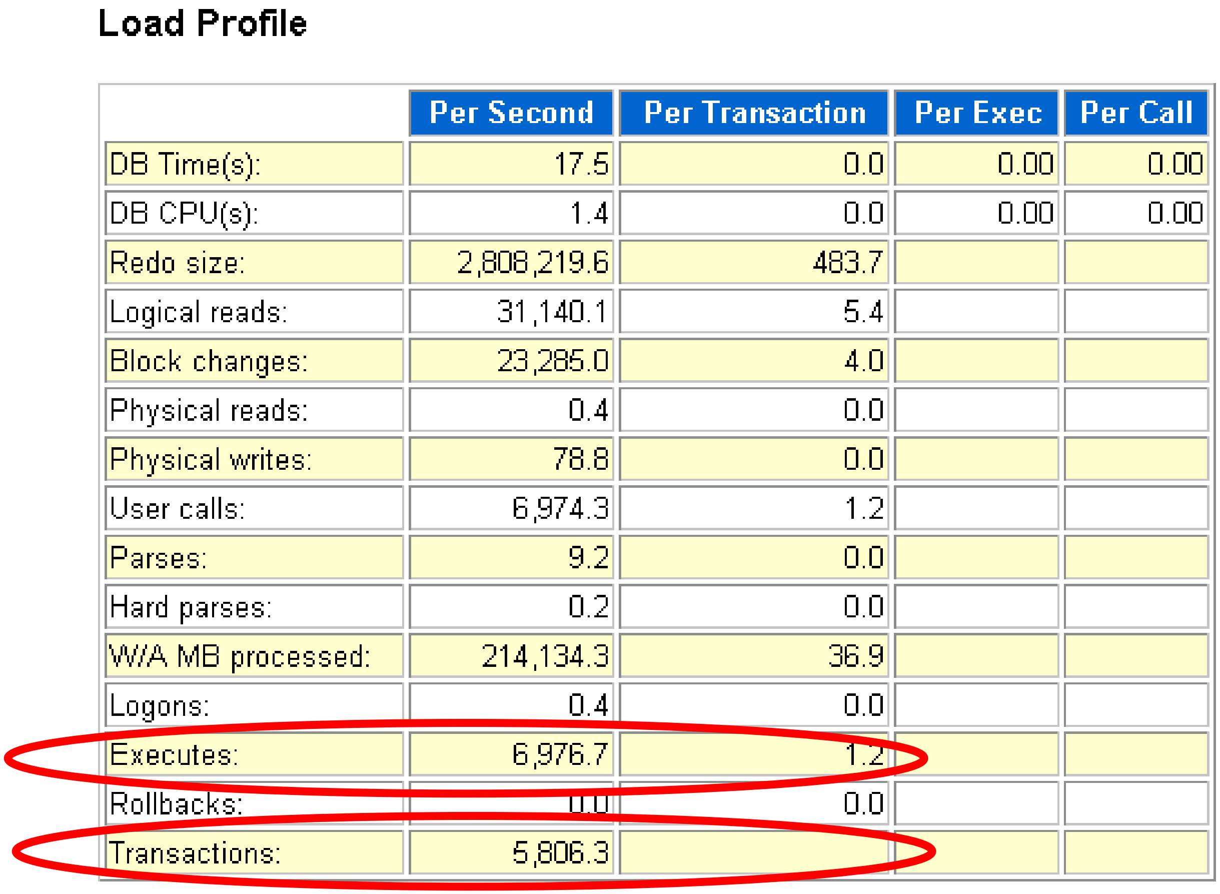
- Use new flavors of prepare/release calls
  - OCIStmtPrepare2(), OCIStmtRelease()

Soft Parsing Best Practices
Session Cached Cursors in the Database

- Fallback if you cannot change the application to use statement caching
- `session_cached_cursors = X`
  - Defaults have changed in various releases
  - Oracle Database 11*g* Default = 50

Wrong Default

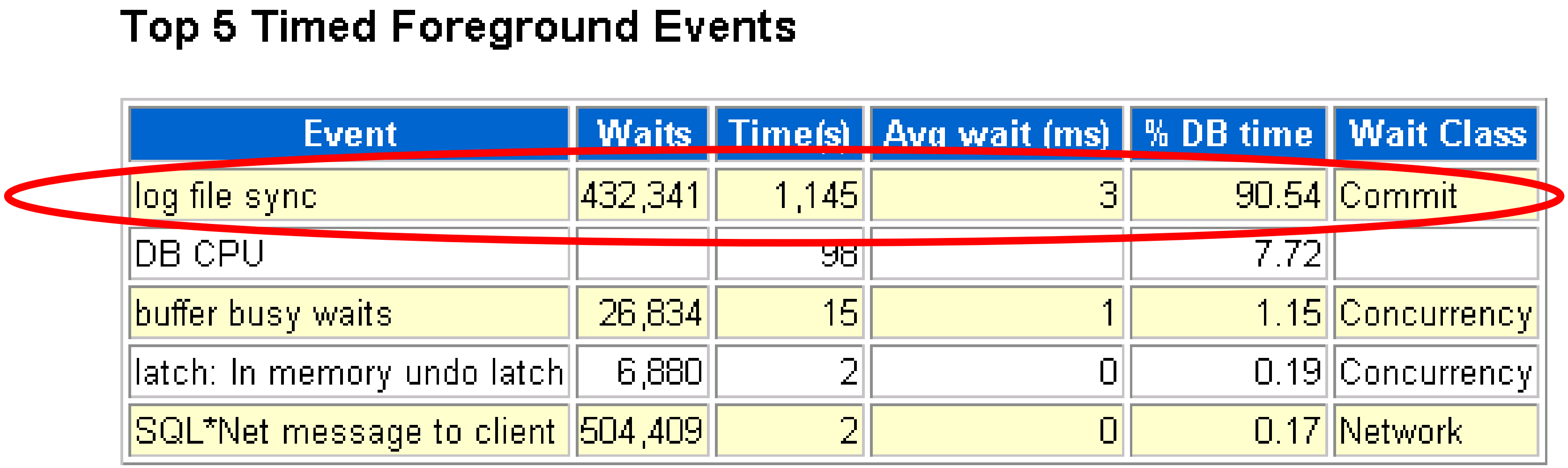
Wrong Default: AWR report



Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	17.5	0.0	0.00	0.00
DB CPU(s):	1.4	0.0	0.00	0.00
Redo size:	2,808,219.6	483.7		
Logical reads:	31,140.1	5.4		
Block changes:	23,285.0	4.0		
Physical reads:	0.4	0.0		
Physical writes:	78.8	0.0		
User calls:	6,974.3	1.2		
Parses:	9.2	0.0		
Hard parses:	0.2	0.0		
W/A MB processed:	214,134.3	36.9		
Logons:	0.4	0.0		
Executes:	6,976.7	1.2		
Rollbacks:	0.0	0.0		
Transactions:	5,806.3			

AWR Report: excessive transaction activity



Top 5 Timed Foreground Events

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
log file sync	432,341	1,145	3	90.54	Commit
DB CPU		98		7.72	
buffer busy waits	26,834	15	1	1.15	Concurrency
latch: In memory undo latch	6,880	2	0	0.19	Concurrency
SQL*Net message to client	504,409	2	0	0.17	Network

Wrong Default: ADDM Recommendations

```
Finding 2: Commits and Rollbacks
Impact is 15.69 active sessions, 90.54% of total activity.
----------------------------------------------------------
Waits on event "log file sync" while performing COMMIT and ROLLBACK operations
were consuming significant database time.

   Recommendation 1: Application Analysis
   Estimated benefit is 15.69 active sessions, 90.54% of total activity.
   ---------------------------------------------------------------------
   Action
      Investigate application logic for possible reduction in the number of
      COMMIT operations by increasing the size of transactions.
   Rationale
      The application was performing 345218 transactions per minute with an
      average redo size of 483 bytes per transaction.
```

Wrong Default
Auto Commits

- Beware. Many database drivers (e.g. JDBC) have auto commit on
  - Causes more transactions, log flushes
  - Increases response time
  - Breaks atomicity of the transactions
- Use driver specific knob to turn off auto commits
  - e.g. JDBC
    - `conn.setAutoCommit(false);`

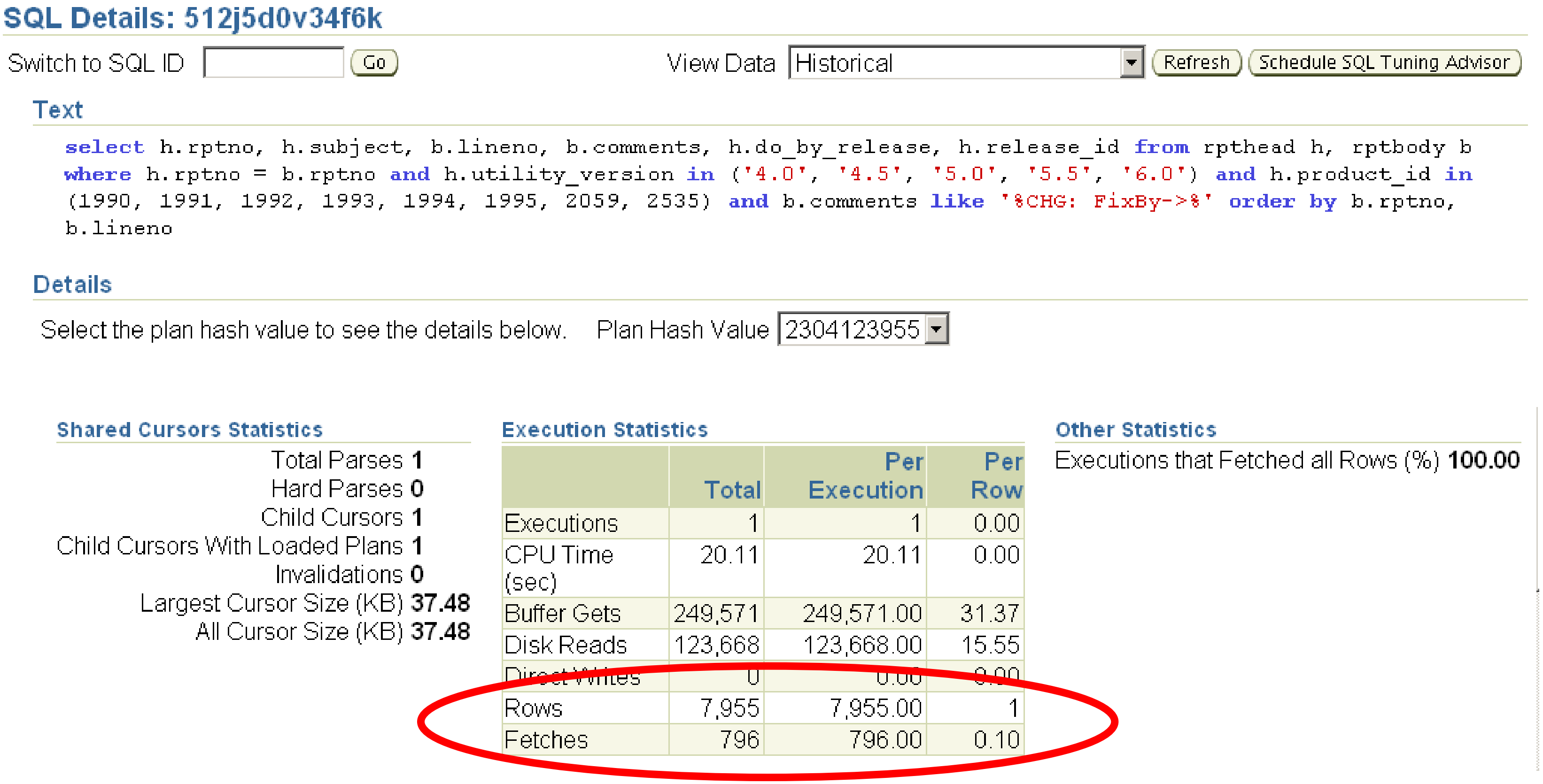
Array DMLs

Array Fetch size from V$SQL example

```
SQL> select sql_text, executions, fetches, rows_processed from V$SQL
     where sql_text like 'select city from locations';

SQL_TEXT                   EXECUTIONS    FETCHES ROWS_PROCESSED
------------------------------ ---------- ---------- --------------
select city from locations     8800      26400     202400
```

- Looking at V$SQL
  - ROWS_PROCESSED/EXECUTION = 23
  - Bump up client side prefetch or array-fetch to 24
  - Fetches all rows in one roundtrip (instead of three)
- V$SQL information can get aged out
  - Same statistics available via persistent AWR tables
  - DBA_HIST_SQLSTAT, DBA_HIST_SQLTEXT

Array Fetch size from Enterprise Manager



SQL Details: 512j5d0v34f6k

Switch to SQL ID [ ] Go View Data Historical Refresh Schedule SQL Tuning Advisor

Text

```
select h.rptno, h.subject, b.lineno, b.comments, h.do_by_release, h.release_id from rpthead h, rptbody b
where h.rptno = b.rptno and h.utility_version in ('4.0', '4.5', '5.0', '5.5', '6.0') and h.product_id in
(1990, 1991, 1992, 1993, 1994, 1995, 2059, 2535) and b.comments like '%CHG: FixBy->%' order by b.rptno,
b.lineno
```

Details

Select the plan hash value to see the details below. Plan Hash Value 2304123955

Shared Cursors Statistics

Total Parses	**1**
Hard Parses	**0**
Child Cursors	**1**
Child Cursors With Loaded Plans	**1**
Invalidations	**0**
Largest Cursor Size (KB)	**37.48**
All Cursor Size (KB)	**37.48**

Execution Statistics

	Total	Per Execution	Per Row
Executions	1	1	0.00
CPU Time (sec)	20.11	20.11	0.00
Buffer Gets	249,571	249,571.00	31.37
Disk Reads	123,668	123,668.00	15.55
Direct Writes	0	0.00	0.00
Rows	7,955	7,955.00	1
Fetches	796	796.00	0.10

Other Statistics

Executions that Fetched all Rows (%) **100.00**

Array Fetching in Java

```
String query = "SELECT EMPLOYEE_ID, LAST_NAME FROM EMPLOYEES "
                      +" WHERE EMPLOYEE_ID > ? "
                      +" ORDER BY EMPLOYEE_ID";

 pstmt = connection.prepareStatement(query);
 pstmt.setInt(1, generateNumber(MIN_EMPLOYEE_ID, MAX_EMPLOYEE_ID));
 pstmt.setFetchSize(20);
 rs = pstmt.executeQuery();
 ResultSetMetaData rsmd = rs.getMetaData();
 int columnCount = rsmd.getColumnCount();
 while (rs.next()) {
    for(int i = 1; i <= columnCount; ++i)
     System.out.println(rsmd.getColumnName(i) +"["
       +rsmd.getColumnTypeName(i) +"]: "
       +rs.getString(i));
 }
```

Array DML in Java

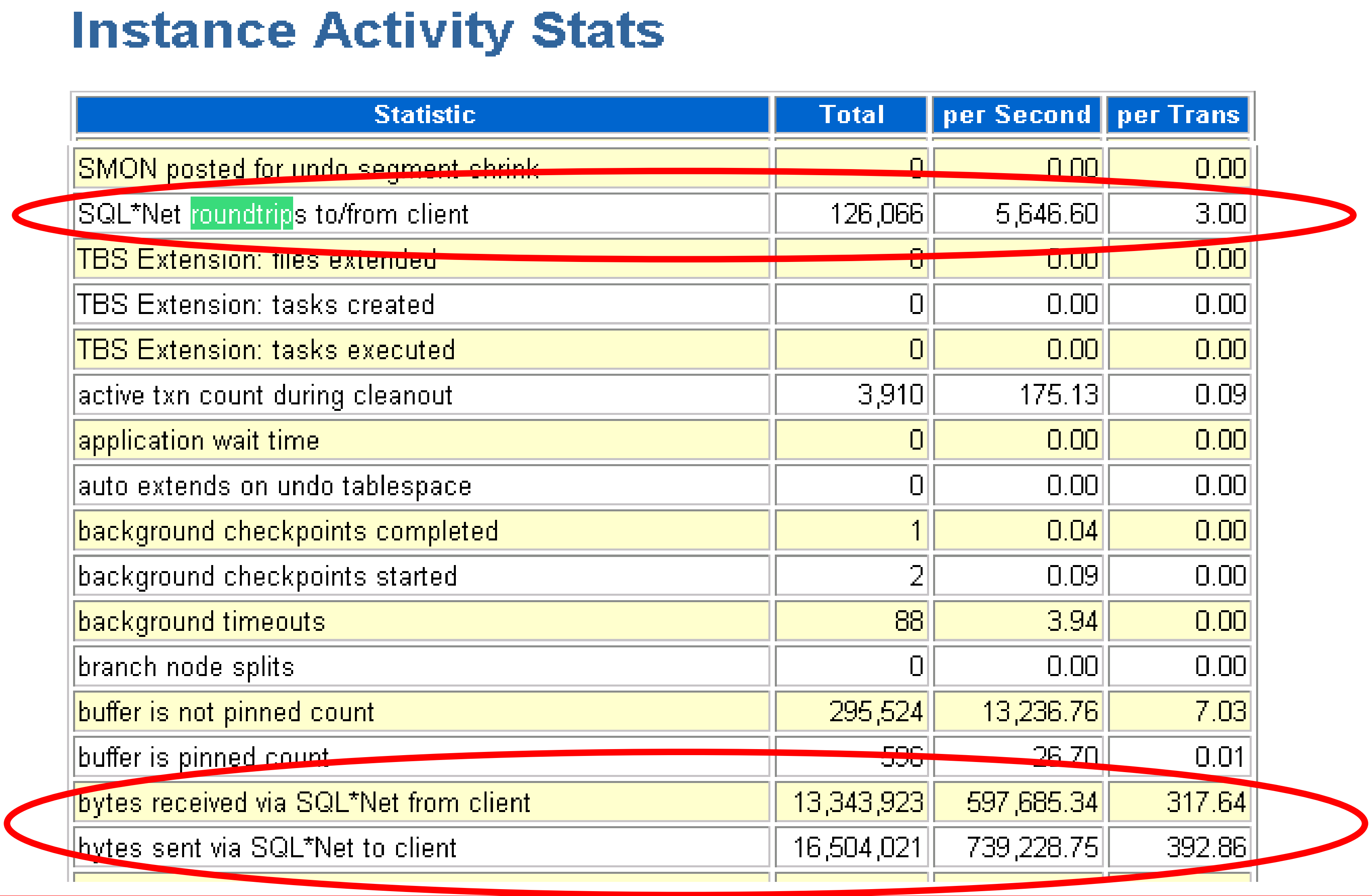
```
String dml = "UPDATE EMPLOYEES SET SALARY = ?"
            +" WHERE EMPLOYEE_ID = ?";
pstmt = connection.prepareStatement(dml);
((OraclePreparedStatement)pstmt).setExecuteBatch(UPDATE_COUNT);
for(int i = 0; i < UPDATE_COUNT; ++i)
{
  pstmt.setInt(1, generateNumber(MIN_SALARY, MAX_SALARY));
  pstmt.setInt(2, generateNumber(min, max));
  pstmt.executeUpdate();
  completedOp++;
}
```

Stored Procedures

Stored Procedures and Best Practices

- Bundle multiple SQL statements in one call
  - Use anonymous blocks or stored procedures
  - Eliminates roundtrips to database
  - Eliminates moving data between database and client
- Can improve performance dramatically
- Monitor roundtrips and bytes transferred stats
  - High values may indicate optimization opportunities
- Oracle furnishes Java and PL/SQL Stored Procedures

Stored Procedures: AWR report



Instance Activity Stats

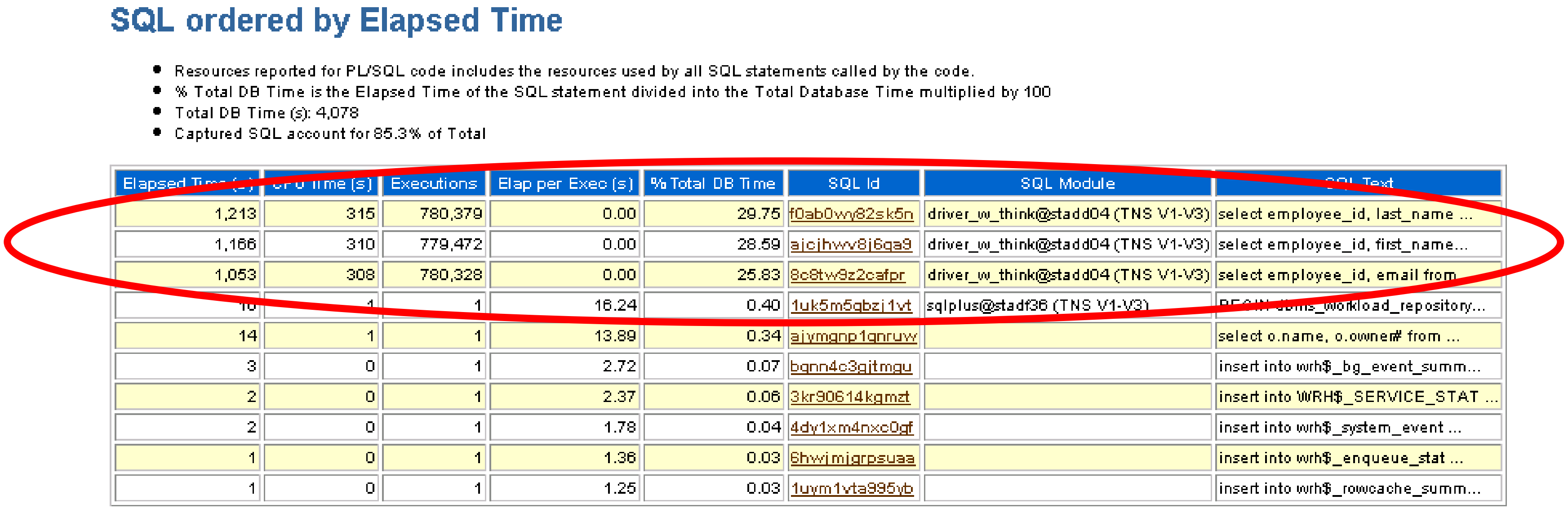
Statistic	Total	per Second	per Trans
SMON posted for undo segment shrink	0	0.00	0.00
SQL*Net roundtrips to/from client	126,066	5,646.60	3.00
TBS Extension: files extended	0	0.00	0.00
TBS Extension: tasks created	0	0.00	0.00
TBS Extension: tasks executed	0	0.00	0.00
active txn count during cleanout	3,910	175.13	0.09
application wait time	0	0.00	0.00
auto extends on undo tablespace	0	0.00	0.00
background checkpoints completed	1	0.04	0.00
background checkpoints started	2	0.09	0.00
background timeouts	88	3.94	0.00
branch node splits	0	0.00	0.00
buffer is not pinned count	295,524	13,236.76	7.03
buffer is pinned count	596	26.70	0.01
bytes received via SQL*Net from client	13,343,923	597,685.34	317.64
bytes sent via SQL*Net to client	16,504,021	739,228.75	392.86

Client-side Result Caching

Identifying Candidate Queries for Client Result Caching from

- Identify top SELECT statements
  - BY CPU
  - BY Elapsed Time
- Pick queries
  - On tables that are not updated often
  - With result sets can fit in available client memory

Identifying Candidate Queries for Client Result Caching from AWR

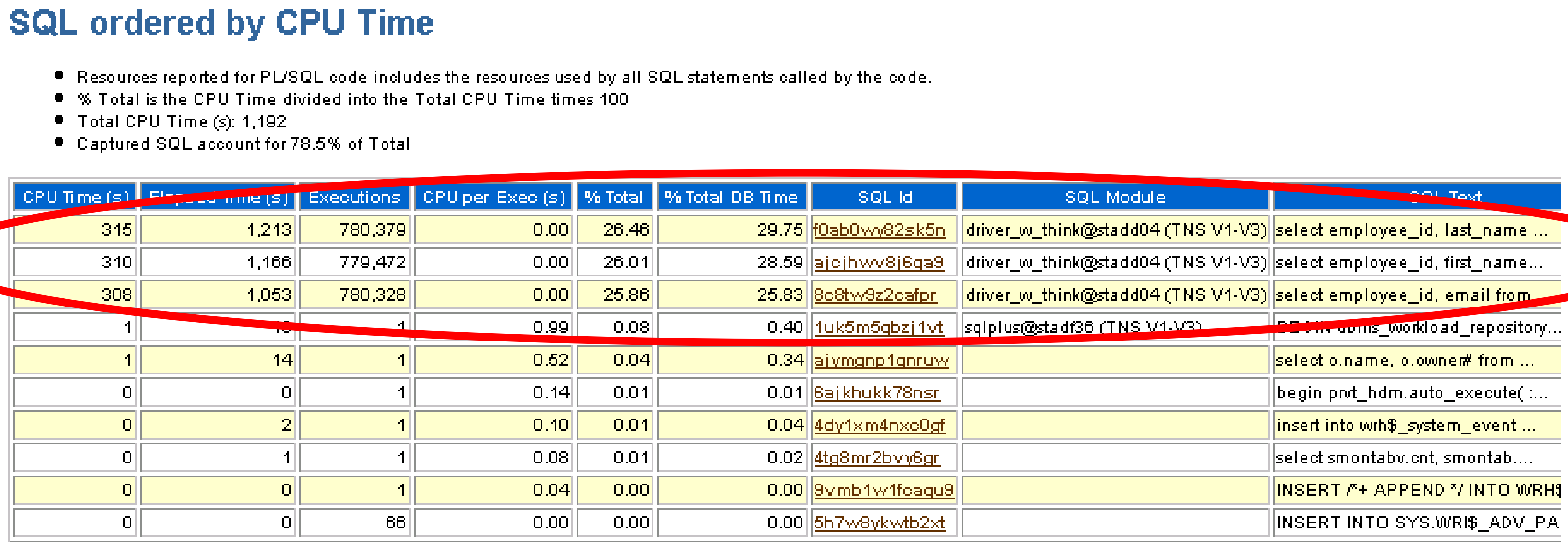


SQL ordered by Elapsed Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total DB Time is the Elapsed Time of the SQL statement divided into the Total Database Time multiplied by 100
- Total DB Time (s): 4,078
- Captured SQL account for 85.3% of Total

Elapsed Time (s)	CPU Time (s)	Executions	Elap per Exec (s)	% Total DB Time	SQL Id	SQL Module	SQL Text
1,213	315	780,379	0.00	29.75	f0ab0wy82sk5n	driver_w_think@stadd04 (TNS V1-V3)	select employee_id, last_name ...
1,166	310	779,472	0.00	28.59	ajcjhwv8j6qa9	driver_w_think@stadd04 (TNS V1-V3)	select employee_id, first_name...
1,053	308	780,328	0.00	25.83	8c8tw9z2cafpr	driver_w_think@stadd04 (TNS V1-V3)	select employee_id, email from
16	1	1	16.24	0.40	1uk5m5qbzj1vt	sqlplus@stadf36 (TNS V1-V3)	BEGIN dbms_workload_repository...
14	1	1	13.89	0.34	ajymgnp1gnruw		select o.name, o.owner# from ...
3	0	1	2.72	0.07	bgnn4c3gjtmgu		insert into wrh$_bg_event_summ...
2	0	1	2.37	0.06	3kr90614kgmzt		insert into WRH$_SERVICE_STAT ...
2	0	1	1.78	0.04	4dy1xm4nxc0gf		insert into wrh$_system_event ...
1	0	1	1.36	0.03	6hwjmjgrpsuaa		insert into wrh$_enqueue_stat ...
1	0	1	1.25	0.03	1uym1vta995yb		insert into wrh$_rowcache_summ...

Identifying Candidate Queries for Client Result Caching from AWR



SQL ordered by CPU Time

- Resources reported for PL/SQL code includes the resources used by all SQL statements called by the code.
- % Total is the CPU Time divided into the Total CPU Time times 100
- Total CPU Time (s): 1,192
- Captured SQL account for 78.5% of Total

CPU Time (s)	Elapsed Time (s)	Executions	CPU per Exec (s)	% Total	% Total DB Time	SQL Id	SQL Module	SQL Text
315	1,213	780,379	0.00	26.46	29.75	f0ab0wy82sk5n	driver_w_think@stadd04 (TNS V1-V3)	select employee_id, last_name ...
310	1,166	779,472	0.00	26.01	28.59	ajcjhwv8j6qa9	driver_w_think@stadd04 (TNS V1-V3)	select employee_id, first_name...
308	1,053	780,328	0.00	25.86	25.83	8c8tw9z2cafpr	driver_w_think@stadd04 (TNS V1-V3)	select employee_id, email from ...
1	16	1	0.99	0.08	0.40	1uk5m5qbzj1vt	sqlplus@stadf36 (TNS V1-V3)	BEGIN dbms_workload_repository...
1	14	1	0.52	0.04	0.34	ajymgnp1gnruw		select o.name, o.owner# from ...
0	0	1	0.14	0.01	0.01	6ajkhukk78nsr		begin prvt_hdm.auto_execute(:...
0	2	1	0.10	0.01	0.04	4dy1xm4nxc0gf		insert into wrh$_system_event ...
0	1	1	0.08	0.01	0.02	4tg8mr2bvy6gr		select smontabv.cnt, smontab....
0	0	1	0.04	0.00	0.00	9vmb1w1fcaqu9		INSERT /*+ APPEND */ INTO WRH$
0	0	66	0.00	0.00	0.00	5h7w8ykwtb2xt		INSERT INTO SYS.WRI$_ADV_PA

Result Set Caching with Oracle Database

- 11gR2: choose tables or view to be cached

  Caching is transparent to the application

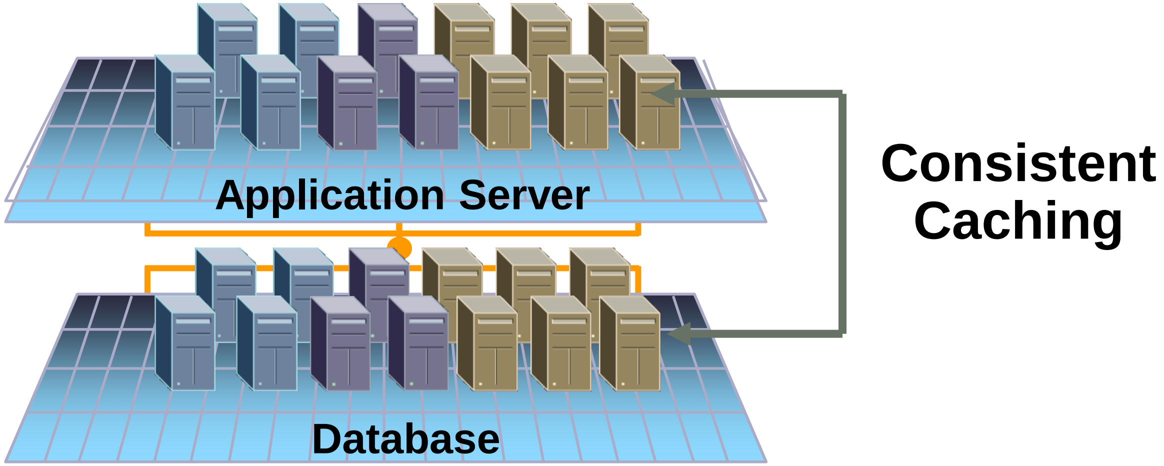
```
   create table sales (...) result_cache

    alter table last_name result_cache

    create view v2 as
       select /*+ result cache */ col1, coln from t1
```

- 11gR1: developer must add hint to the SQL query

```
select /*+ result_cache */ last_name from employees
```

Transparent Client-side Result Set Cache

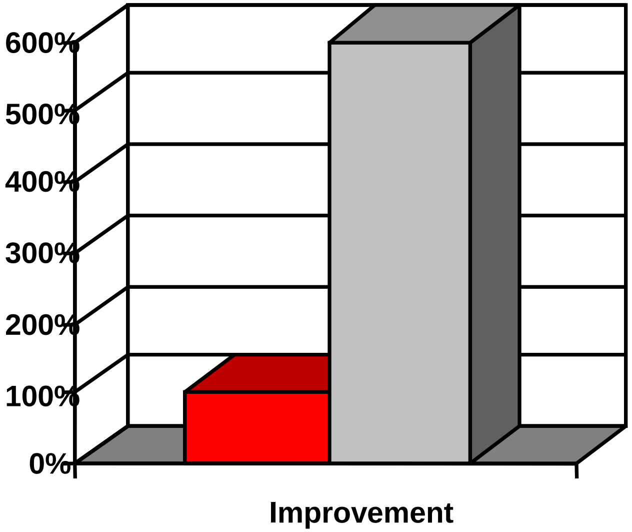
- The Query Results Set is Cached on the client-side
- Cache Consistency is maintained by the driver (using Query Change Notification

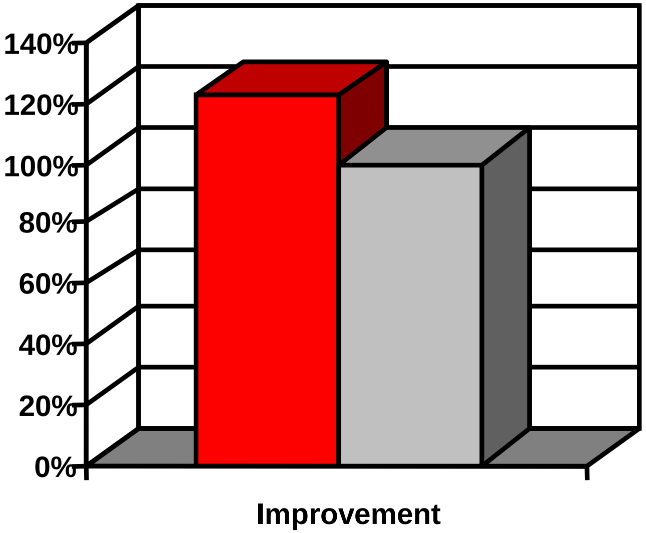


- init.ora parameter

  CLIENT_RESULT_CACHE_SIZE

Niles Benchmark Performance Improvements





DB CPU Reduction:

Up to
600%

Response Time :

Up to
15-22% Faster

LOBs

LOBs and Best Practices

- LOB API
  - Recommended for offset based access
  - Use for large LOBs (MBs)
  - Extra roundtrips (pre 11*g*) to get data, length, chunk-size
- Data API
  - Handle LOBs like LONG or LONG RAW columns
  - Recommended for small LOBs
  - No extra roundtrips
- Oracle Database 11*g* Improvements for LOBs
  - BASIC LOBs: Tune SDU & Use PreFetching
  - SECUREFILES LOBs: Vectored I/O (a.k.a. Zero Copy network transfer)

BASIC LOBs

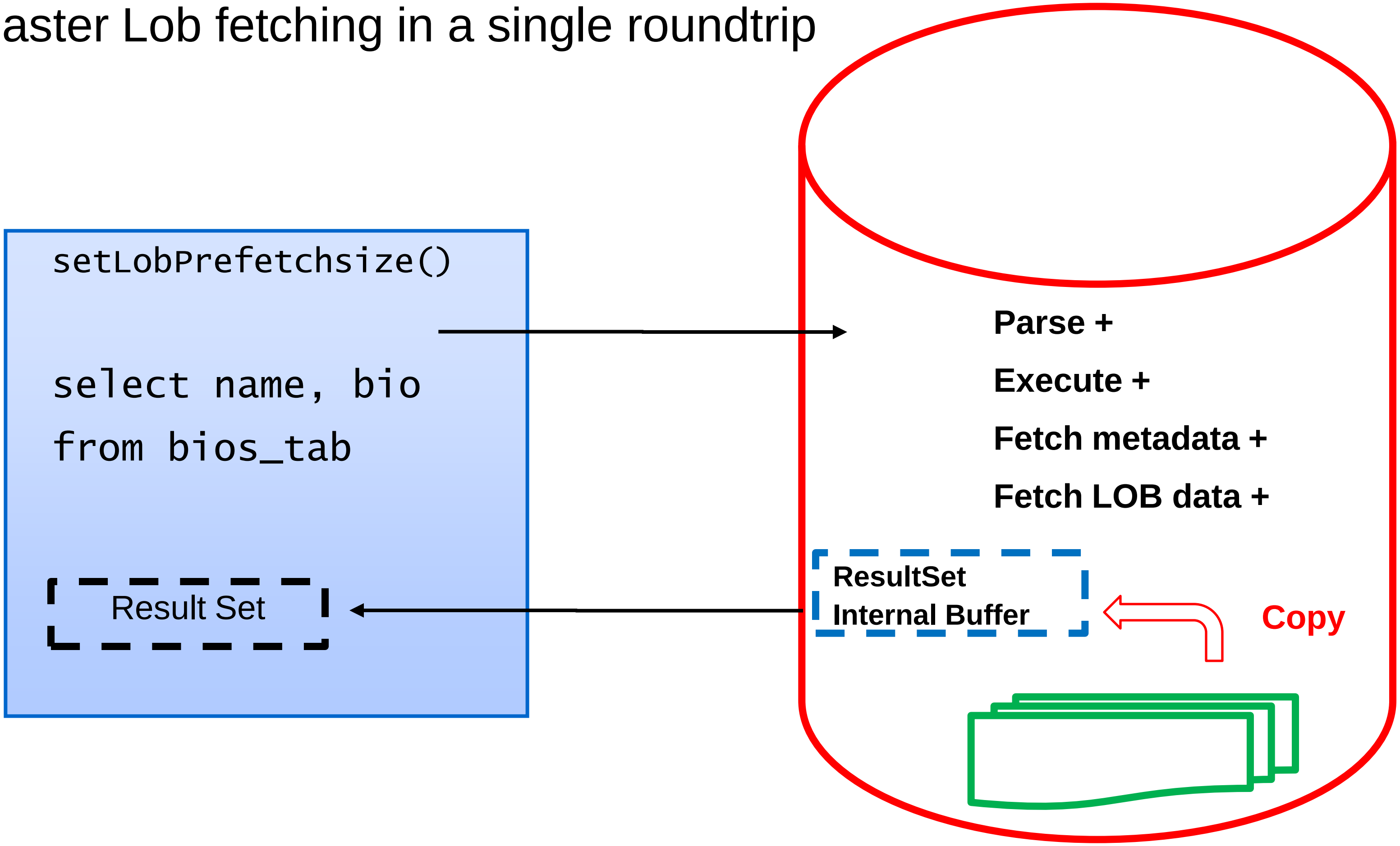
Optimize SDU_SIZE for Large Data Transfers

- Controls SQL*Net packet size
- Default is 8k starting with Oracle Database 11*g*
- Set it upto 64k (with Oracle 11*g*R2) if application does
  - Large Result set array fetches
  - Large Array DML operations
  - Large PL/SQL IN/OUT bind transfers
  - Needs to be set on both client and server
- Monitor network stats in AWR

BASIC LOB

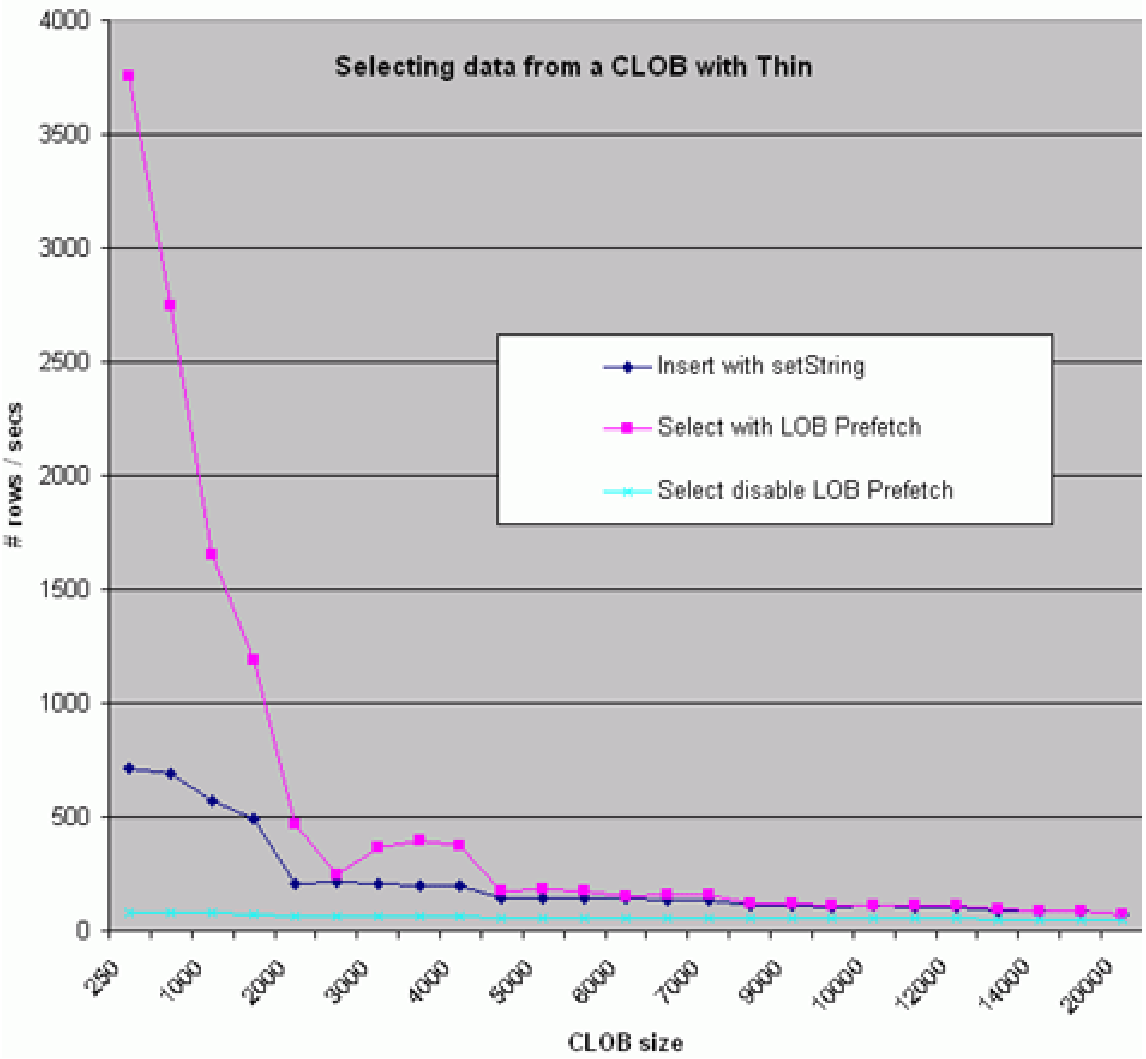
LOB Prefetching

Faster Lob fetching in a single roundtrip

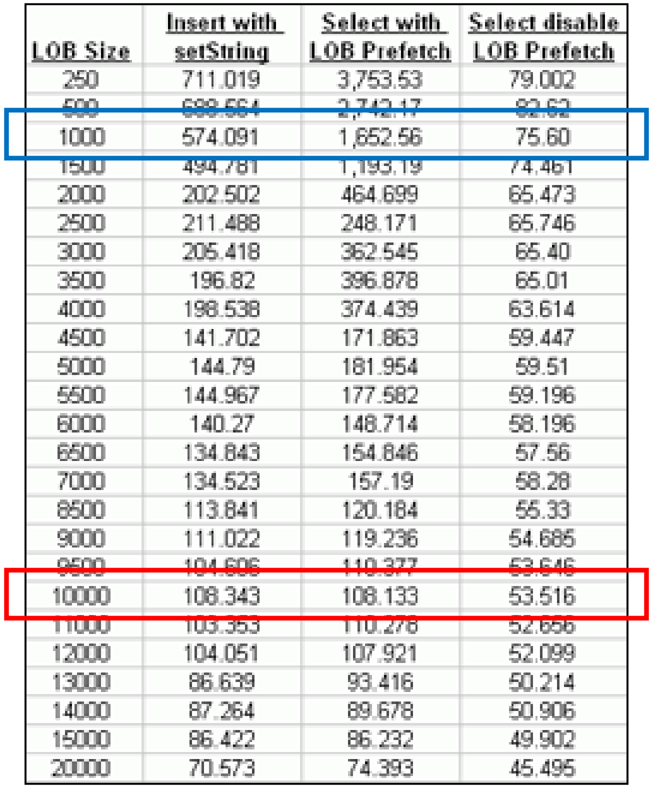


```
setLobPrefetchsize()

select name, bio
from bios_tab
```

LOB PreFetching Performance



Throughput (per sec)

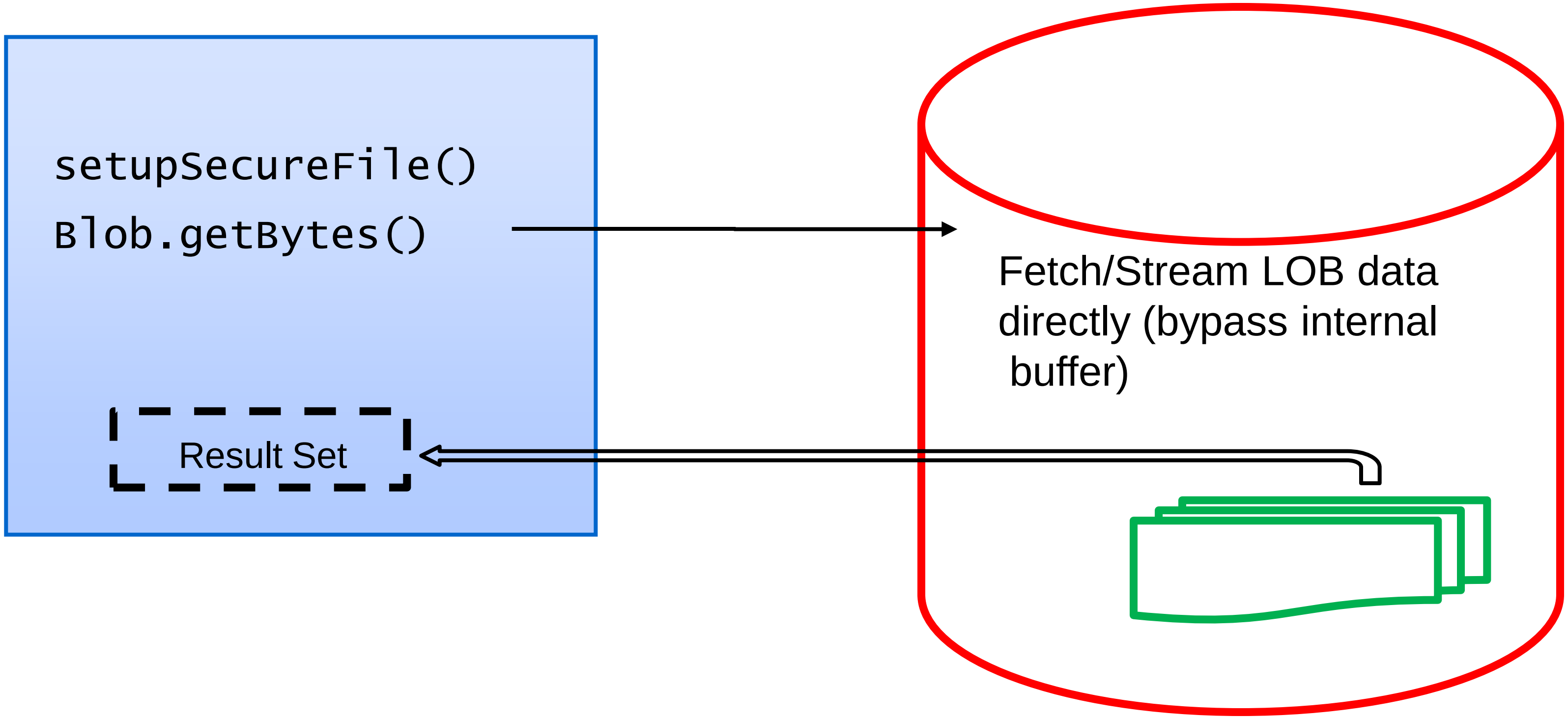


LOB Size	Insert with setString	Select with LOB Prefetch	Select disable LOB Prefetch
250	711.019	3,753.53	79.002
500	688.564	2,742.17	82.62
1000	574.091	1,652.56	75.60
1500	494.781	1,193.19	74.461
2000	202.502	464.699	65.473
2500	211.488	248.171	65.746
3000	205.418	362.545	65.40
3500	196.82	396.878	65.01
4000	198.538	374.439	63.614
4500	141.702	171.863	59.447
5000	144.79	181.954	59.51
5500	144.967	177.582	59.196
6000	140.27	148.714	58.196
6500	134.843	154.846	57.56
7000	134.523	157.19	58.28
8500	113.841	120.184	55.33
9000	111.022	119.236	54.685
9500	104.606	110.377	53.646
10000	108.343	108.133	53.516
11000	103.353	110.278	52.656
12000	104.051	107.921	52.099
13000	86.639	93.416	50.214
14000	87.264	89.678	50.906
15000	86.422	86.232	49.902
20000	70.573	74.393	45.495

SecureFiles LOBs
Optimize Very Large LOBs operations

Large Reads/Writes

- BASIC LOBs: internal buffer copy are expensive
- SECUREFILE LOBS: *“Zero-copy IO” or “Vectored i/o mechanism”*



```
setupSecureFile()
Blob.getBytes()
```

Application Development Best Practices

- Connection Pooling
- Bind Variables
- Statement Caching
- Turn off Auto Commits
- Reducing Roundtrips
  - Array DML
  - Array Fetching and Prefetching
  - PL/SQL and Java stored procedures
- Stored Procedures
- Result Caching
- LOBs/Secure Files

White Paper

- Building High Performance Drivers for Oracle Database 11*g*: OCI Tips and Techniques
  - www.oracle.com/technology/tech/oci/pdf/building-best-drivers.v9.pdf

Q & A

Hardware and Software

ORACLE

Engineered to Work Together

ORACLE®