Deutsche Bank
Identifier

Anatomy of cursor

some aspect of cursors management in Oracle

PGDays 2017

About me

- 15+ years experience in Oracle databases development/administrating. OCE Oracle SQL.
- MSSQL Server and Sybase experience.
- Use to be a good Java background ☺
- Author of a couple well-known web projects (kontramarka.ru -1st version, sonystyle.ru - 1st version, …)
- My blog: https://dmitryremizov.wordpress.com/

Deutsche Bank
Identifier

Why?

- We use Oracle database as a backend for Oracle Coherence.
  (Coherence is a distributed in-memory cache)

- What is Coherence from the database standpoint?

Data access patterns are dictated by Coherence interfaces

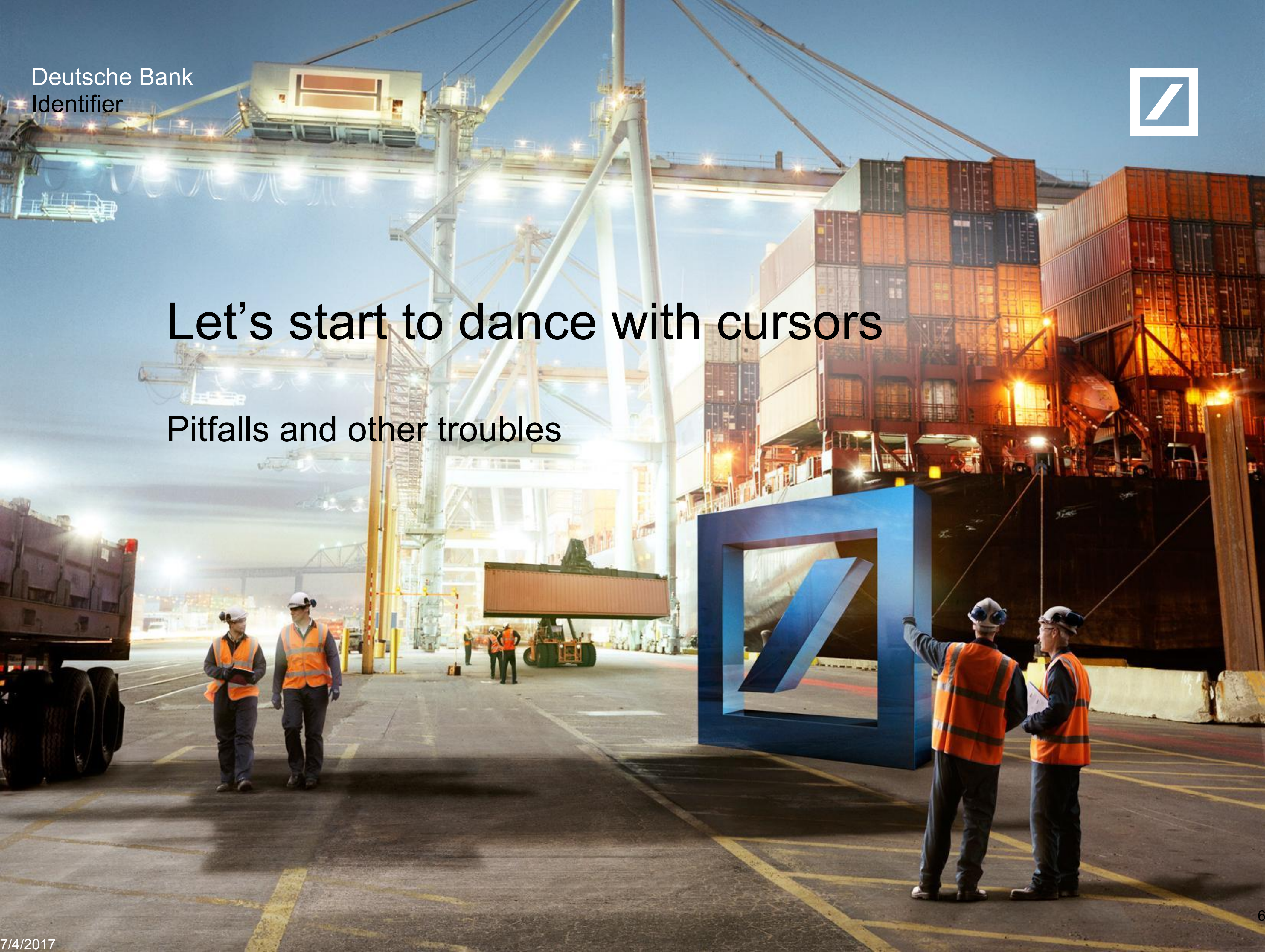
CacheStore Interface extends CacheLoader	On Oracle side
Object load(Object var1);	SELECT * FROM T WHERE id = ?
Map loadAll(Collection var1);	SELECT * FROM T WHERE id in (?, ?,..
void store(Object var1, Object var2);	INSERT INTO T (…) VALUES (?,?,…
void storeAll(Map var1);	INSERT INTO T (…) VALUES (?,?,… (with batchUpdate)
void erase(Object var1);	-- we don’t use it
void eraseAll(Collection var1);	--we don’t use it

Data access patterns are dictated by data volumes

- NO DELETEs
- NO UPDATEs
- NO FK
- A very limited usage of Unique Keys

- Best delete is insert, best update is insert again ☺

- We prefer to lost a bit of CPU on more complex SELECT (almost each entity has a sequence_number column and we usually choose the last version of truth by the sequence number).



Let’s start to dance with cursors

Pitfalls and other troubles

Cursor definition (official ☺)
for just in case

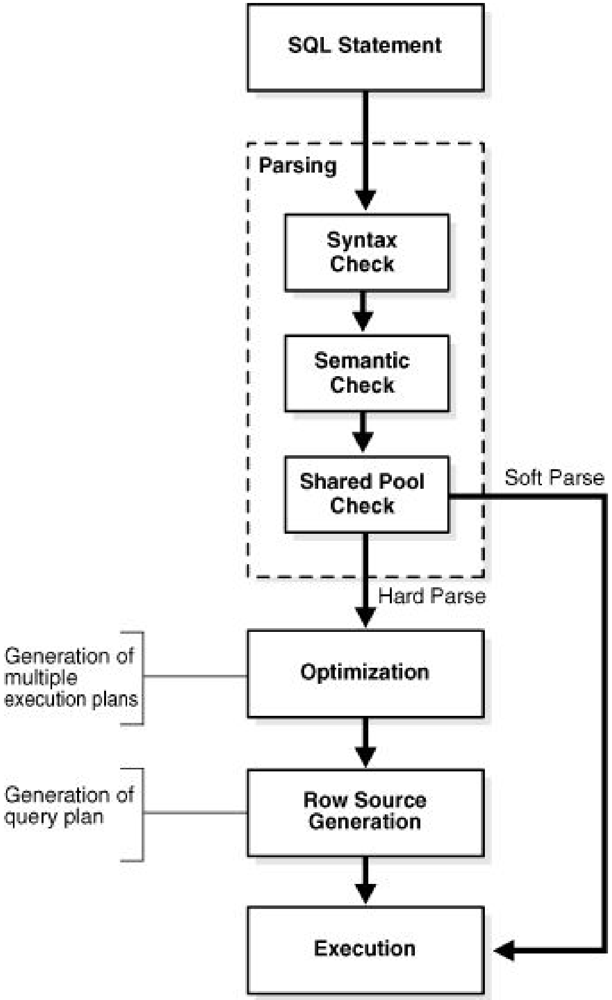
cursor

An area in memory that holds a parsed statement and other information for processing. The cursor/private SQL area contains data such as **bind variable** values, **query** execution state information, and query execution work areas.

child cursor

The cursor containing the plan, compilation environment, and other information for a statement whose text is stored in a parent cursor.

SQL Processing steps



Ideal case

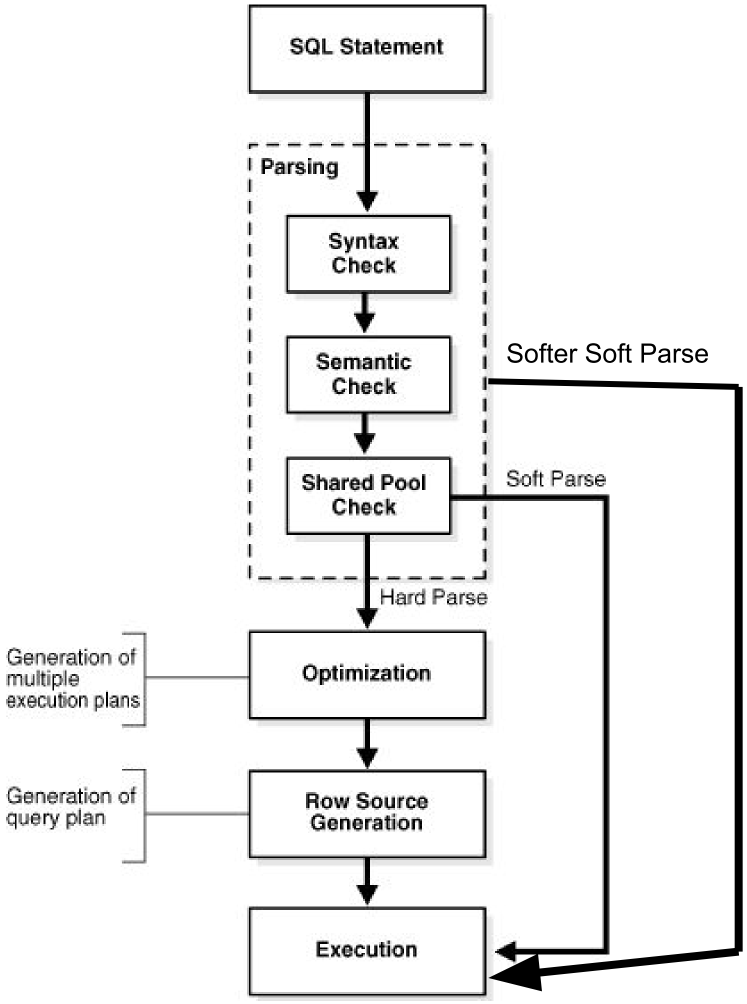
No Parse Execution

There are approximately 4 “types” of SQL cursor execution in Oracle.

- hard parse
- soft parse
- softer soft parse
- no parse

Useful links
https://alexzeng.wordpress.com/2012/12/31/oracle-core-hard-parse-soft-parse-soft-soft-parse-no-parse-at-all/

SQL Processing steps



Soft parse steps

Syntax Check

Oracle Database must check each SQL statement for syntactic validity. A statement that breaks a rule for well-formed SQL syntax fails the check. For example, the following statement fails because the keyword FROM is misspelled as FORM:

```
SQL> SELECT * FORM employees;

SELECT * FORM employees
     *
ERROR at line 1:

ORA-00923: FROM keyword not found where expected
```

Semantic Check

The semantics of a statement are its meaning. Thus, a semantic check determines whether a statement is meaningful, for example, whether the objects and columns in the statement exist. A syntactically correct statement can fail a semantic check, as shown in the following example of a query of a nonexistent table:

```
SQL> SELECT * FROM nonexistent_table;

SELECT * FROM nonexistent_table
      *
ERROR at line 1:
ORA-00942: table or view does not exist
```

Shared Pool Check

During the parse, the database performs a shared pool check to determine whether it can skip resource-intensive steps of statement processing. To this end, the database uses a hashing algorithm to generate a hash value for every SQL statement. The statement hash value is the SQL ID shown in V$SQL.SQL_ID.

When a user submits a SQL statement, the database searches the shared SQL area to see if an existing parsed statement has the same hash value. The hash value of a SQL statement is distinct from the following values:

Memory address for the statement

What can be and can’t be “No parse” logically

You need to pass something during “No parse” execution

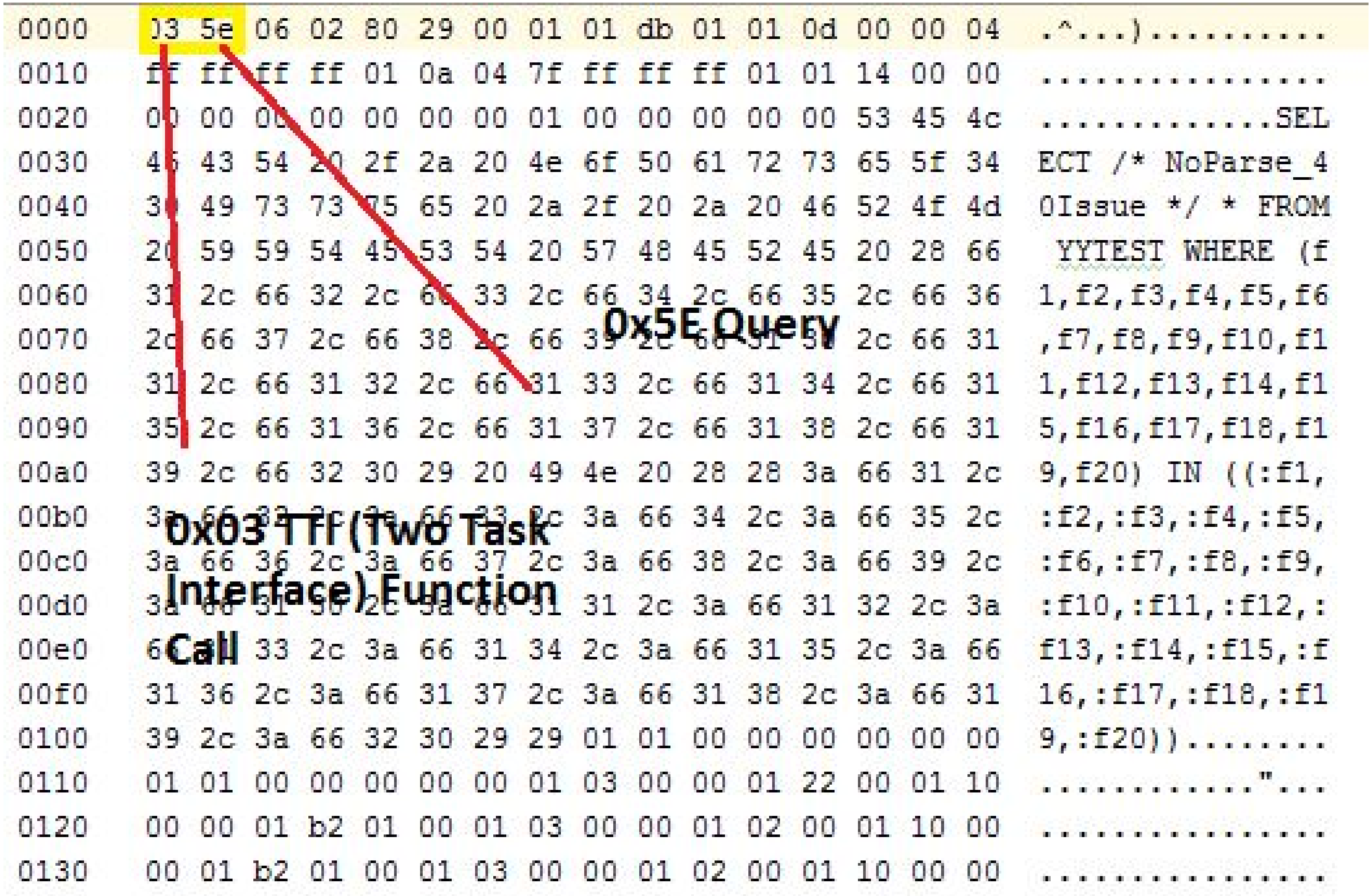
- It can’t be SQL text itself obviously.
- You need to pass some identified between client and server side but what exactly
  - sql_id ?
  - Some internal address ?
  - Something else?

JDBC test case

```
 Connection con = dataSource.getConnection();

PreparedStatement ps = con.prepareStatement(SQL);
PreparedStatement ps2 = con.prepareStatement(SQL2);
-- No network roundtrip so far
for …………….. {
--------------------some PreparedStatement bindings----------------------------
     for …………….. {
     ResultSet rs= ps.executeQuery();
          --network roundtrip is here:
     ResultSet rs2= ps2.executeQuery();
     -----------------------some ResultSet reading part----------------
     rs.close();
     rs2.close();
-- No network roundtrip so far
```

First execution



```
0000    03 5e 06 02 80 29 00 01 01 db 01 01 0d 00 00 04   .^...)..........
0010    ff ff ff ff 01 0a 04 7f ff ff ff 01 01 14 00 00   ................
0020    00 00 00 00 00 00 00 01 00 00 00 00 00 53 45 4c   .............SEL
0030    45 43 54 20 2f 2a 20 4e 6f 50 61 72 73 65 5f 34   ECT /* NoParse_4
0040    30 49 73 73 75 65 20 2a 2f 20 2a 20 46 52 4f 4d   0Issue */ * FROM
0050    20 59 59 54 45 53 54 20 57 48 45 52 45 20 28 66    YYTEST WHERE (f
0060    31 2c 66 32 2c 66 33 2c 66 34 2c 66 35 2c 66 36   1,f2,f3,f4,f5,f6
0070    2c 66 37 2c 66 38 2c 66 39 2c 66 31 30 2c 66 31   ,f7,f8,f9,f10,f1
0080    31 2c 66 31 32 2c 66 31 33 2c 66 31 34 2c 66 31   1,f12,f13,f14,f1
0090    35 2c 66 31 36 2c 66 31 37 2c 66 31 38 2c 66 31   5,f16,f17,f18,f1
00a0    39 2c 66 32 30 29 20 49 4e 20 28 28 3a 66 31 2c   9,f20) IN ((:f1,
00b0    3a 66 32 2c 3a 66 33 2c 3a 66 34 2c 3a 66 35 2c   :f2,:f3,:f4,:f5,
00c0    3a 66 36 2c 3a 66 37 2c 3a 66 38 2c 3a 66 39 2c   :f6,:f7,:f8,:f9,
00d0    3a 66 31 30 2c 3a 66 31 31 2c 3a 66 31 32 2c 3a   :f10,:f11,:f12,:
00e0    66 31 33 2c 3a 66 31 34 2c 3a 66 31 35 2c 3a 66   f13,:f14,:f15,:f
00f0    31 36 2c 3a 66 31 37 2c 3a 66 31 38 2c 3a 66 31   16,:f17,:f18,:f1
0100    39 2c 3a 66 32 30 29 29 01 01 00 00 00 00 00 00   9,:f20))........
0110    01 01 00 00 00 00 00 01 03 00 00 01 22 00 01 10   ............"...
0120    00 00 01 b2 01 00 01 03 00 00 01 02 00 01 10 00   ................
0130    00 01 b2 01 00 01 03 00 00 01 02 00 01 10 00 00   ................
```

0x5E Query

0x03 TTI (Two Task Interface) Function Call

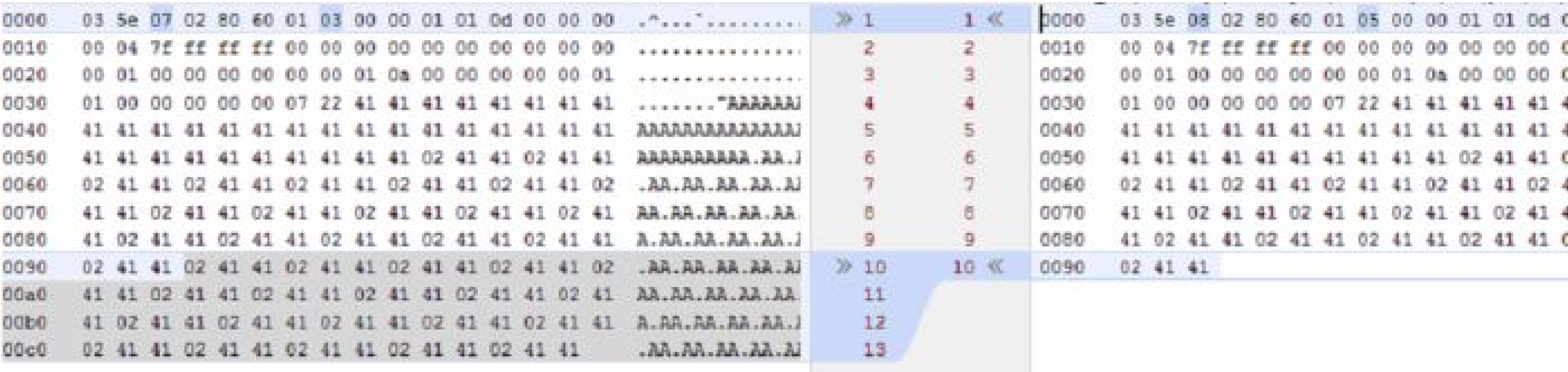
Useful links
http://thesprawl.org/research/oracle-tns-protocol/

0x03	•TTI (Two-Task Interface) Function call. The exact function id comes immediately after data packet id. Below is a table of different TTI IDs:0x02 Open •0x03 Query •0x04 Execute •0x05 Fetch •0x08 Close •0x09 Disconnect/logoff •0x0C AutoCommit ON •0x0D AutoCommit OFF •0x0E Commit •0x0F Rollback •0x14 Cancel •0x2B Describe •0x30 Startup •0x31 Shutdown •0x3B Version •0x43 K2 Transactions •0x47 Query •0x4A OSQL7 •0x5C OKOD •0x5E Query •0x60 LOB Operations •0x62 ODNY •0x67 Transaction - end •0x68 Transaction - begin •0x69 OCCA •0x6D Startup •0x51 Logon (present password) •0x52 Logon (present username) •0x73 Logon (present password - send AUTH_PASSWORD) •0x76 Logon (present username - request AUTH_SESSKEY) •0x77 Describe •0x7F OOTCM •0x8B OKPFC

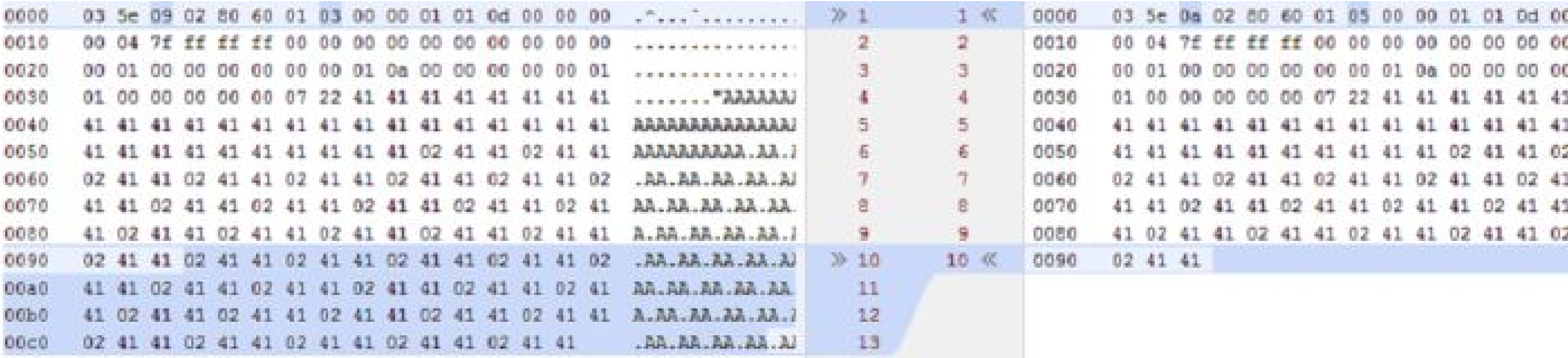
Useful links

http://thesprawl.org/research/oracle-tns-protocol/

2nd and 3rd execution (“No parse”)



```
0000   03 5e 07 02 80 60 01 03 00 00 01 01 0d 00 00 00  .^...`.........   » 1    1 «   0000   03 5e 08 02 80 60 01 05 00 00 01 01 0d
0010   00 04 7f ff ff ff 00 00 00 00 00 00 00 00 00 00  ................     2    2     0010   00 04 7f ff ff ff 00 00 00 00 00 00 00
0020   00 01 00 00 00 00 00 00 01 0a 00 00 00 00 00 01  ................     3    3     0020   00 01 00 00 00 00 00 00 01 0a 00 00 00
0030   01 00 00 00 00 00 07 22 41 41 41 41 41 41 41 41  ......."AAAAAAJ     4    4     0030   01 00 00 00 00 00 07 22 41 41 41 41 41
0040   41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAJ     5    5     0040   41 41 41 41 41 41 41 41 41 41 41 41 41
0050   41 41 41 41 41 41 41 41 41 41 02 41 41 02 41 41  AAAAAAAAAA.AA.J     6    6     0050   41 41 41 41 41 41 41 41 41 41 02 41 41
0060   02 41 41 02 41 41 02 41 41 02 41 41 02 41 41 02  .AA.AA.AA.AA.AJ     7    7     0060   02 41 41 02 41 41 02 41 41 02 41 41 02
0070   41 41 02 41 41 02 41 41 02 41 41 02 41 41 02 41  AA.AA.AA.AA.AA.     8    8     0070   41 41 02 41 41 02 41 41 02 41 41 02 41
0080   41 02 41 41 02 41 41 02 41 41 02 41 41 02 41 41  A.AA.AA.AA.AA.J     9    9     0080   41 02 41 41 02 41 41 02 41 41 02 41 41
0090   02 41 41 02 41 41 02 41 41 02 41 41 02 41 41 02  .AA.AA.AA.AA.AJ   » 10  10 «   0090   02 41 41
00a0   41 41 02 41 41 02 41 41 02 41 41 02 41 41 02 41  AA.AA.AA.AA.AA.     11
00b0   41 02 41 41 02 41 41 02 41 41 02 41 41 02 41 41  A.AA.AA.AA.AA.J     12
00c0   02 41 41 02 41 41 02 41 41 02 41 41 02 41 41     .AA.AA.AA.AA.AJ     13
```



```
0000   03 5e 09 02 80 60 01 03 00 00 01 01 0d 00 00 00  .^...`.........   » 1    1 «   0000   03 5e 0a 02 80 60 01 05 00 00 01 01 0d 00
0010   00 04 7f ff ff ff 00 00 00 00 00 00 00 00 00 00  ................     2    2     0010   00 04 7f ff ff ff 00 00 00 00 00 00 00 00
0020   00 01 00 00 00 00 00 00 01 0a 00 00 00 00 00 01  ................     3    3     0020   00 01 00 00 00 00 00 00 01 0a 00 00 00 00
0030   01 00 00 00 00 00 07 22 41 41 41 41 41 41 41 41  ......."AAAAAAJ     4    4     0030   01 00 00 00 00 00 07 22 41 41 41 41 41 41
0040   41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41  AAAAAAAAAAAAAAJ     5    5     0040   41 41 41 41 41 41 41 41 41 41 41 41 41 41
0050   41 41 41 41 41 41 41 41 41 41 02 41 41 02 41 41  AAAAAAAAAA.AA.J     6    6     0050   41 41 41 41 41 41 41 41 41 41 02 41 41 02
0060   02 41 41 02 41 41 02 41 41 02 41 41 02 41 41 02  .AA.AA.AA.AA.AJ     7    7     0060   02 41 41 02 41 41 02 41 41 02 41 41 02 41
0070   41 41 02 41 41 02 41 41 02 41 41 02 41 41 02 41  AA.AA.AA.AA.AA.     8    8     0070   41 41 02 41 41 02 41 41 02 41 41 02 41 41
0080   41 02 41 41 02 41 41 02 41 41 02 41 41 02 41 41  A.AA.AA.AA.AA.J     9    9     0080   41 02 41 41 02 41 41 02 41 41 02 41 41 02
0090   02 41 41 02 41 41 02 41 41 02 41 41 02 41 41 02  .AA.AA.AA.AA.AJ   » 10  10 «   0090   02 41 41
00a0   41 41 02 41 41 02 41 41 02 41 41 02 41 41 02 41  AA.AA.AA.AA.AA.     11
00b0   41 02 41 41 02 41 41 02 41 41 02 41 41 02 41 41  A.AA.AA.AA.AA.J     12
00c0   02 41 41 02 41 41 02 41 41 02 41 41 02 41 41     .AA.AA.AA.AA.AJ     13
```

WHAT does it mean all those number

03 or 05 from previous slide

Let’s do a CURSORDUMP as described here:
http://blog.tanelpoder.com/2009/07/09/identify-the-sql-statement-causing-those-wait-x-lines-in-a-top-truncated-sql-tracefile/

```
oradebug setospid 1820
oradebug dump cursordump 1
```

And finally we found

```
Cursor#3(0xffff80ffbde11c48) state=BOUND curiob=0xffff80ffbde28a40
 curflg=4e    0    bd8db0
----- Dump    1t xsc=0xffff80ffbde28a40
cur=0xffff80
LibraryHandle:  Address=11b2d5ae8 Hash=569dbc39 LockMode=N PinMode=0
LoadLockMode=0 Status=VALD
  ObjectName:  Name=SELECT /* NoParse_40Issue */ * FROM YYTEST
WHERE
(f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,f16,f17,f18,f19,f20,f21,f22,f23,f24,
f25,f26,f27,f28,f29,f30,f31,f32,f33,f34,f35,f36,f37,f38,f39,f40) IN
((:f1,:f2,:f3,:f4,:f5,:f6,:f7,:f8,:f9,:f10,:f11,:f12,:f13,:f14,:f15,:f16,:f17,:f18,:f19,:f20,:f
21,:f22,:f23,:f24,:f25,:f26,:f27,:f28,:f29,:f30,:f31,:f32,:f33,:f34,:f35,:f36,:f37,:f38,:f
39,:f40))
    FullHashValue=

**************************************************************************
```

slot number in open cursor array in UGA

And one more confirmation

Cursor#5(0xffff80ffbde11d68) state=BOUND curiob=0xffff80ffbdca1f80

curflg=4e 0 d8db0

----- Dump za xsc=0xffff80ffbdca1f80 cur=0xffff80

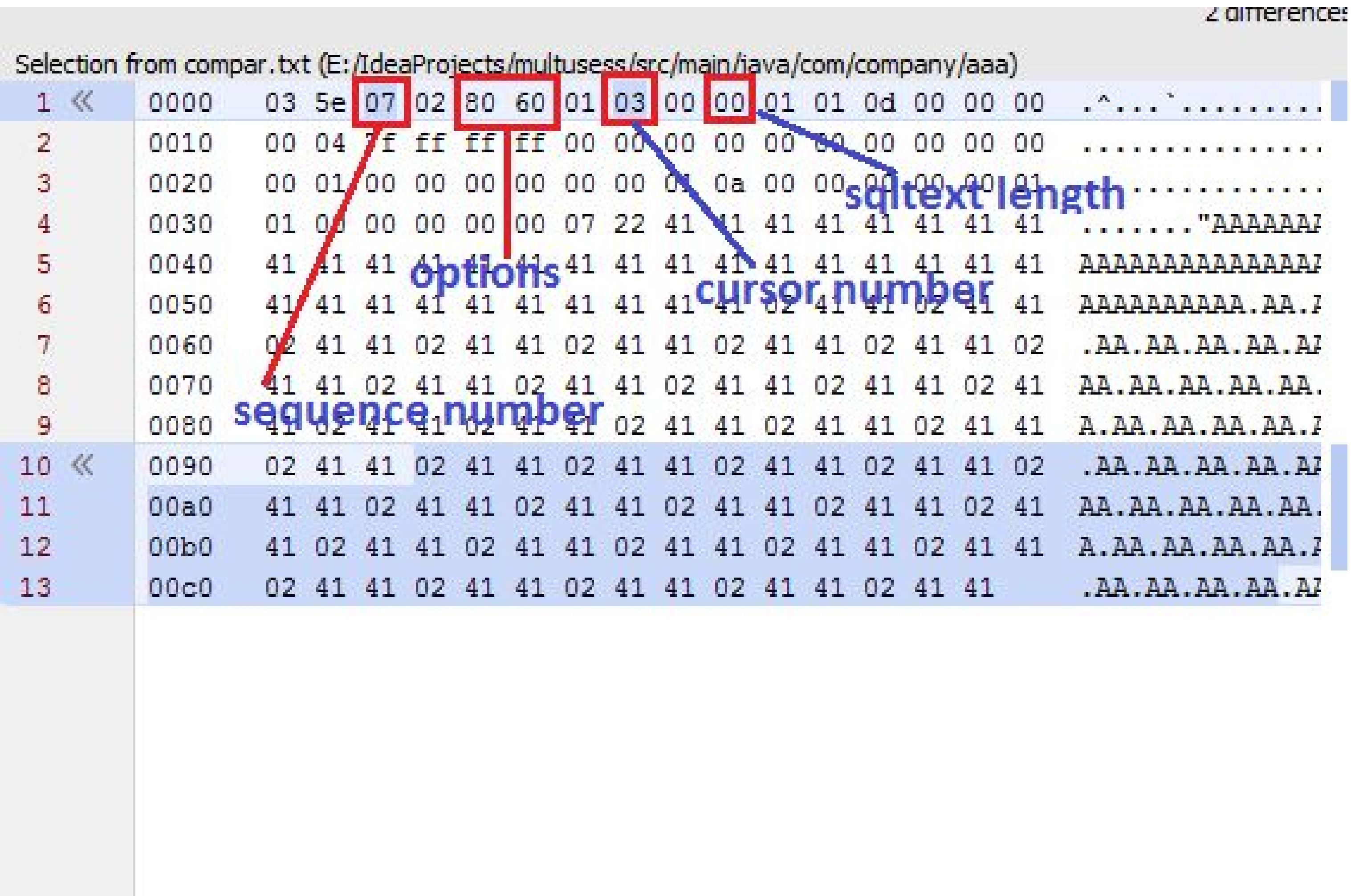
slot number in open cursor array in UGA

LibraryHandle: Address=1fb2ce130 Hash=fc058fea LockMode=N PinMode=0 LoadLockMode=0 Status=VALD

ObjectName: Name=SELECT /* NoParse_40Issue */ * FROM YYTEST WHERE (f1,f2,f3,f4,f5,f6,f7,f8,f9,f10,f11,f12,f13,f14,f15,f16,f17,f18,f19,f20) IN ((:f1,:f2,:f3,:f4,:f5,:f6,:f7,:f8,:f9,:f10,:f11,:f12,:f13,:f14,:f15,:f16,:f17,:f18,:f19,:f20))

FullHashValue=fa2532489a58f35a824f9964fc058fea

Bonus pack ☺



2 differences

Selection from compar.txt (E:/IdeaProjects/multusess/src/main/java/com/company/aaa)

```
0000    03 5e 07 02 80 60 01 03 00 00 01 01 0d 00 00 00   .^...`.........
0010    00 04 7f ff ff ff 00 00 00 00 00 00 00 00 00 00   ...............
0020    00 01 00 00 00 00 00 00 01 0a 00 00 00 00 00 01   ...............
0030    01 00 00 00 00 00 07 22 41 41 41 41 41 41 41 41   ......."AAAAAAA
0040    41 41 41 41 41 41 41 41 41 41 41 41 41 41 41 41   AAAAAAAAAAAAAAA
0050    41 41 41 41 41 41 41 41 41 41 02 41 41 02 41 41   AAAAAAAAAA.AA.A
0060    02 41 41 02 41 41 02 41 41 02 41 41 02 41 41 02   .AA.AA.AA.AA.AA
0070    41 41 02 41 41 02 41 41 02 41 41 02 41 41 02 41   AA.AA.AA.AA.AA.
0080    41 02 41 41 02 41 41 02 41 41 02 41 41 02 41 41   A.AA.AA.AA.AA.A
0090    02 41 41 02 41 41 02 41 41 02 41 41 02 41 41 02   .AA.AA.AA.AA.AA
00a0    41 41 02 41 41 02 41 41 02 41 41 02 41 41 02 41   AA.AA.AA.AA.AA.
00b0    41 02 41 41 02 41 41 02 41 41 02 41 41 02 41 41   A.AA.AA.AA.AA.A
00c0    02 41 41 02 41 41 02 41 41 02 41 41 02 41 41      .AA.AA.AA.AA.AA
```

How to achieve “No Parse”

```
<bean id="dataSource" class="oracle.jdbc.pool.OracleDataSource" destroy-method="close">
<property name="URL" value="jdbc:oracle:thin:@KJGHJKGHKGKG.db.com:1521/XXXXX.de.db.com"/>
…………………………………………………………………….
……………………………………………………………………..
    <property name="connectionProperties">
        <props>
            <prop key="v$session.program">insConn</prop>
        </props>
    </property>
    <property name="connectionCacheProperties">
        <props>
            <prop key="MinLimit">1</prop>
            <prop key="MaxLimit">1</prop>
            <prop key="InitialLimit">1</prop>
            <prop key="MaxStatementsLimit">100</prop>
        </props>
    </property>
</bean>

OR

((OracleDataSource) dataSource).setConnectionProperties(properties);
```

Cursors issues and pitfalls

- Shared pool abuse
  - Subpools
  - Untyped binds issue
  - Binds size issue
  - Hard parse without parse
- DML concurrency and related issues
  - Spring auto-commit

Shared pool abuse
(subpools)

You have to have quite a lot of cursors when deal with Coherence cluster.

Lets calculate:

500 different entities * 500 different binds = ~25000 parent cursors.

You have to multiply it to 10-20 for child cursors per parent cursor

(the reason/s will be provided on the next slides).

That gives us ~ 0.25 - 0.5 mln of different cursors, in reality it would order of magnitude less but still a lot.

According to Oracle Corp. support recommendations we had to set : _kghdsidx_count =1 to avoid a severe shared pool fragmentation(that is a number of shared pool subpools by the way).

Useful links
https://andreynikolaev.wordpress.com/

Workaround for too many cursors

Instead of having:

```
SELECT … FROM T where ID in (?);      --1 bind placeholders
SELECT … FROM T where ID in (?,?);   --2
SELECT … FROM T where ID in (?,?,?);      --3
SELECT … FROM T where ID in (?,?,?,?);   --4
SELECT … FROM T where ID in (?,?,?,?,?);--5
SELECT … FROM T where ID in (?,?,?,?,?,?);   --6
SELECT … FROM T where ID in (?,?,?,?,?,?,?); --7
SELECT … FROM T where ID in (?,?,?,?,?,?,?,?);    --8
```

Do

```
SELECT … FROM T where ID in (?);      --1
SELECT … FROM T where ID in (?,?);   --2
SELECT … FROM T where ID in (?,?,?,?);   --4
SELECT … FROM T where ID in (?,?,?,?,?,?,?,?); --8
```

i.e. 1,2,4,8,16,…,512 bind placeholders and populate last binds with the last value.

“Untyped” bind variables issues

Test case for “untyped” binds

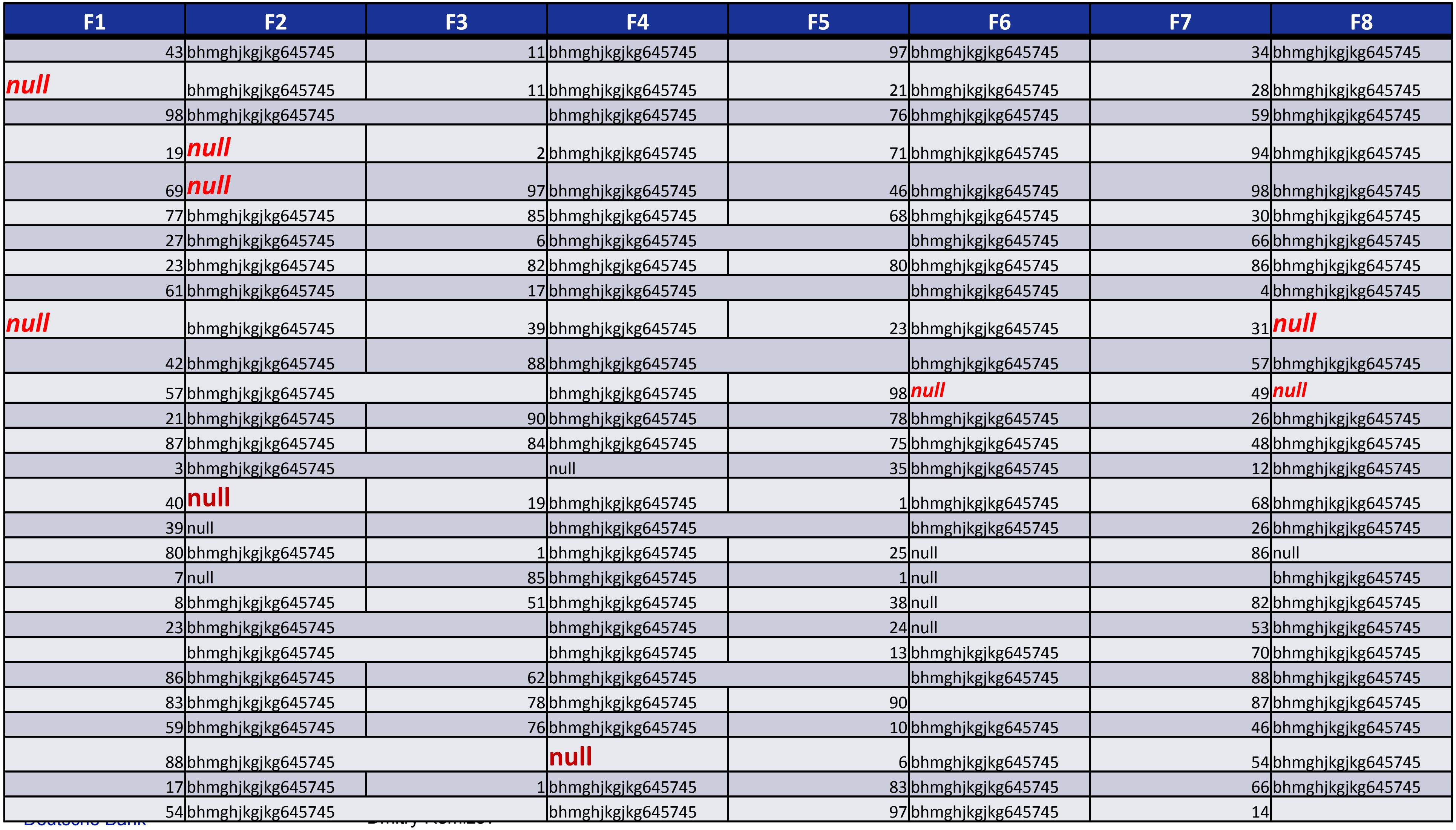
We will do some insert in a table of the following structure:

```
CREATE TABLE YYTEST(
    F1 NUMBER,
    F2 VARCHAR2(255),
    F3 NUMBER,
    F4 VARCHAR2(255),
    F5 NUMBER,
    F6 VARCHAR2(255),
    F7 NUMBER,
    F8 VARCHAR2(255)
)
```

On the next page I’ll show an approx. dataset to insert.

Data generation description.

i.e. more or less evenly distributed **nulls** across the dataset



F1	F2	F3	F4	F5	F6	F7	F8
43	bhmghjkgjkg645745	11	bhmghjkgjkg645745	97	bhmghjkgjkg645745	34	bhmghjkgjkg645745
null	bhmghjkgjkg645745	11	bhmghjkgjkg645745	21	bhmghjkgjkg645745	28	bhmghjkgjkg645745
98	bhmghjkgjkg645745		bhmghjkgjkg645745	76	bhmghjkgjkg645745	59	bhmghjkgjkg645745
19	null	2	bhmghjkgjkg645745	71	bhmghjkgjkg645745	94	bhmghjkgjkg645745
69	null	97	bhmghjkgjkg645745	46	bhmghjkgjkg645745	98	bhmghjkgjkg645745
77	bhmghjkgjkg645745	85	bhmghjkgjkg645745	68	bhmghjkgjkg645745	30	bhmghjkgjkg645745
27	bhmghjkgjkg645745	6	bhmghjkgjkg645745		bhmghjkgjkg645745	66	bhmghjkgjkg645745
23	bhmghjkgjkg645745	82	bhmghjkgjkg645745	80	bhmghjkgjkg645745	86	bhmghjkgjkg645745
61	bhmghjkgjkg645745	17	bhmghjkgjkg645745		bhmghjkgjkg645745	4	bhmghjkgjkg645745
null	bhmghjkgjkg645745	39	bhmghjkgjkg645745	23	bhmghjkgjkg645745	31	null
42	bhmghjkgjkg645745	88	bhmghjkgjkg645745		bhmghjkgjkg645745	57	bhmghjkgjkg645745
57	bhmghjkgjkg645745		bhmghjkgjkg645745	98	null	49	null
21	bhmghjkgjkg645745	90	bhmghjkgjkg645745	78	bhmghjkgjkg645745	26	bhmghjkgjkg645745
87	bhmghjkgjkg645745	84	bhmghjkgjkg645745	75	bhmghjkgjkg645745	48	bhmghjkgjkg645745
3	bhmghjkgjkg645745		null	35	bhmghjkgjkg645745	12	bhmghjkgjkg645745
40	null	19	bhmghjkgjkg645745	1	bhmghjkgjkg645745	68	bhmghjkgjkg645745
39	null		bhmghjkgjkg645745		bhmghjkgjkg645745	26	bhmghjkgjkg645745
80	bhmghjkgjkg645745	1	bhmghjkgjkg645745	25	null	86	null
7	null	85	bhmghjkgjkg645745	1	null		bhmghjkgjkg645745
8	bhmghjkgjkg645745	51	bhmghjkgjkg645745	38	null	82	bhmghjkgjkg645745
23	bhmghjkgjkg645745		bhmghjkgjkg645745	24	null	53	bhmghjkgjkg645745
	bhmghjkgjkg645745		bhmghjkgjkg645745	13	bhmghjkgjkg645745	70	bhmghjkgjkg645745
86	bhmghjkgjkg645745	62	bhmghjkgjkg645745		bhmghjkgjkg645745	88	bhmghjkgjkg645745
83	bhmghjkgjkg645745	78	bhmghjkgjkg645745	90		87	bhmghjkgjkg645745
59	bhmghjkgjkg645745	76	bhmghjkgjkg645745	10	bhmghjkgjkg645745	46	bhmghjkgjkg645745
88	bhmghjkgjkg645745		null	6	bhmghjkgjkg645745	54	bhmghjkgjkg645745
17	bhmghjkgjkg645745	1	bhmghjkgjkg645745	83	bhmghjkgjkg645745	66	bhmghjkgjkg645745
54	bhmghjkgjkg645745		bhmghjkgjkg645745	97	bhmghjkgjkg645745	14	

“Untyped” bind variables issue. Java code snippet

(you can see resulting dataset on the next slide)

```
SimpleJdbcTemplate simpleJdbcTemplate = new SimpleJdbcTemplate(dataSource);

List<Object[]> batchArgs = new ArrayList<Object[]>();

for (int i = 1; i <= 1000; i++) {
    Map<String, Object> map = new HashMap<String, Object>();
    Object[] item = new Object[8];
    item[0] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[1] = getRandomInt() % 10 == 5 ? null : getRandomString();
    item[2] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[3] = getRandomInt() % 10 == 5 ? null : getRandomString();
    item[4] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[5] = getRandomInt() % 10 == 5 ? null : getRandomString();
    item[6] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[7] = getRandomInt() % 10 == 5 ? null : getRandomString();
    batchArgs.add(item);
}

long startTime = System.currentTimeMillis();
simpleJdbcTemplate.batchUpdate("INSERT /* BatchIssueSpring */ INTO YYYTEST (f1,f2,f3,f4,f5,f6,f7,f8)
VALUES (:f1,:f2,:f3,:f4,:f5,:f6,:f7,:f8)", batchArgs);
System.out.println("time " + ((double) (System.currentTimeMillis() - startTime)) / 1000);
```

We randomly add nulls

Overall timing is ~30 sec

Java code snippet – more correct way

```
SimpleJdbcTemplate simpleJdbcTemplate = new SimpleJdbcTemplate(dataSource);

List<Object[]> batchArgs = new ArrayList<Object[]>();

for (int i = 1; i <= 1000; i++) {
    Map<String, Object> map = new HashMap<String, Object>();
    Object[] item = new Object[8];
    item[0] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[1] = getRandomInt() % 10 == 5 ? null : getRandomString();
    item[2] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[3] = getRandomInt() % 10 == 5 ? null : getRandomString();
    item[4] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[5] = getRandomInt() % 10 == 5 ? null : getRandomString();
    item[6] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[7] = getRandomInt() % 10 == 5 ? null : getRandomString();
    batchArgs.add(item);
}
final int[] argTypes = new int[]{Types.INTEGER, Types.VARCHAR,Types.INTEGER, Types.VARCHAR,
Types.INTEGER, Types.VARCHAR,Types.INTEGER, Types.VARCHAR};

long startTime = System.currentTimeMillis();
simpleJdbcTemplate.batchUpdate("INSERT /* BatchIssueSpring */ INTO YYYTEST (f1,f2,f3,f4,f5,f6,f7,f8)
VALUES (:f1,:f2,:f3,:f4,:f5,:f6,:f7,:f8)", batchArgs, argTypes);
System.out.println("time " + ((double) (System.currentTimeMillis() - startTime)) / 1000);
```

Overall timing is ~0.483 sec (almost 100 times less)

“Untyped” bind variables issue.
Dangerous operations

(applied to Spring framework but very likely the same for various OR mappers)

```
/**
 * Execute a batch using the supplied SQL statement with the batch of supplied arguments.
 * Uses sql with the standard '?' placeholders for parameters
 * @param sql the SQL statement to execute
 * @param batchArgs the List of Object arrays containing the batch of arguments for the query
 * @return an array containing the numbers of rows affected by each update in the batch
 */
public int[] batchUpdate(String sql, List<Object[]> batchArgs);

/**
 * Executes a batch using the supplied SQL statement with the batch of supplied arguments.
 * Uses sql with the named parameter support.
 * @param sql the SQL statement to execute
 * @param batchValues the array of Maps containing the batch of arguments for the query
 * @return an array containing the numbers of rows affected by each update in the batch
 */
public int[] batchUpdate(String sql, Map<String, ?>[] batchValues);
```

“Untyped” bind variables issue.
Correct operations
(requires a bit more typing)

```
/**
 * Execute a batch using the supplied SQL statement with the batch of supplied arguments.
 * Uses sql with the standard '?' placeholders for parameters
 * @param sql the SQL statement to execute.
 * @param batchArgs the List of Object arrays containing the batch of arguments for the query
 * @param argTypes SQL types of the arguments
 * (constants from <code>java.sql.Types</code>)
 * @return an array containing the numbers of rows affected by each update in the batch
 */
public int[] batchUpdate(String sql, List<Object[]> batchArgs, int[] argTypes);
```

```
/**
 * Execute a batch using the supplied SQL statement with the batch of supplied arguments.
 * Uses sql with the named parameter support.
 * @param sql the SQL statement to execute
 * @param batchArgs the array of {@link SqlParameterSource} containing the batch of arguments
 * for the query
 * @return an array containing the numbers of rows affected by each update in the batch
 */
public int[] batchUpdate(String sql, SqlParameterSource[] batchArgs);
```

How it looks like from the database perspective.

SQL cursors for bad case

```
SELECT child_number,
bind_mismatch from
v$sql_shared_cursor where
sql_id='3fnzpd6arjz52';
```

CHILD_NUMBER	BIND_MISMATCH
0	N
1	Y
2	Y
3	Y
4	Y
5	Y
6	Y
7	Y
8	Y
9	Y
10	Y
11	Y
12	Y
13	Y
14	N
15	Y
16	Y
17	Y
18	Y
19	Y
20	N

```
SELECT * from
(SELECT c.CHILD_NUMBER,
c.NAME,c.DATATYPE_STRING from
v$sql_bind_capture c where sql_id='3fnzpd6arjz52'
)
PIVOT (MAX(DATATYPE_STRING) as MX FOR
name IN (':F1',':F2',':F3',':F4',':F5',':F6',':F7',':F8') )
order by 1;
```

CHILD_NUMBER	':F1'_MX	':F2'_MX	':F3'_MX	':F4'_MX	':F5'_MX	':F6'_MX	':F7'_MX	':F8'_MX
0	NUMBER	VARCHAR2(128)	NUMBER	VARCHAR2(128)	NUMBER	VARCHAR2(128)	NUMBER	VARCHAR2(128)
1	**VARCHAR2(32)**	VARCHAR2(128)	NUMBER	VARCHAR2(128)	NUMBER	VARCHAR2(128)	NUMBER	VARCHAR2(128)
2	NUMBER	VARCHAR2(128)	**VARCHAR2(32)**	VARCHAR2(128)	NUMBER	VARCHAR2(128)	NUMBER	VARCHAR2(128)
3	NUMBER	VARCHAR2(128)	NUMBER	VARCHAR2(128)	**VARCHAR2(32)**	VARCHAR2(128)	NUMBER	VARCHAR2(128)
4	NUMBER	VARCHAR2(128)	NUMBER	VARCHAR2(128)	NUMBER	VARCHAR2(128)	**VARCHAR2(32)**	VARCHAR2(128)
5	NUMBER	VARCHAR2(128)	VARCHAR2(32)	VARCHAR2(128)	VARCHAR2(32)	VARCHAR2(128)	NUMBER	VARCHAR2(128)

Untyped binds - root cause/s of bad performance.

Good case		Bad case	
SQL*Net roundtrips to/from client	**3**	SQL*Net roundtrips to/from client	**553**
parse count (total)	**17**	parse count (total)	**566**
parse count (hard)	**0**	parse count (hard)	**0**
user call	**6**	user calls	**555**
user commits	**1**	user commits	**550**

Untyped binds
root cause/s of bad performance.

JDBC driver splits batch to small pieces

Good case

SQL*Net roundtrips to/from client	3
parse count (total)	17
parse count (hard)	0
user calls	6

Bad case

SQL*Net roundtrips to/from client	**553**
parse count (total)	**566**
parse count (hard)	0
user calls	**555**

Untyped binds
root cause/s of bad performance.

Good case

SQL*Net roundtrips to/from client	3
parse count (total)	17
parse count (hard)	0
user calls	6

Bad case

SQL*Net roundtrips to/from client	**553**
parse count (total)	**566**
parse count (hard)	0
user calls	**555**

JDBC driver splits batch to small pieces

Each piece is parsed (at least by soft parse)

Untyped binds
root cause/s of bad performance.

Good case

SQL*Net roundtrips to/from client	3
parse count (total)	17
parse count (hard)	0
user calls	6
user commits	**1**

Bad case

SQL*Net roundtrips to/from client	**553**
parse count (total)	**566**
parse count (hard)	1
user calls	**555**
user commits	**550**

JDBC driver splits batch to small pieces

Each piece is parsed (at least by soft parse)

Under some circumstances it also commits each piece, will be discussed later.

Root cause
from Java/Spring perspective

```
package org.springframework.jdbc.core;

public abstract class StatementCreatorUtils { …..
………
private static void setNull(PreparedStatement ps, int paramIndex, int sqlType, String
typeName)   throws SQLException {

        if (sqlType == SqlTypeValue.TYPE_UNKNOWN) {
                                        ……………………………
                ps.setNull(paramIndex, sqlType);
            }
```

*So we bind our nulls as **SqlTypeValue.TYPE_UNKNOWN** if we don't specify exact SQL type, and this interprets by Oracle as VARCHAR2 bind.*

For someone who wants to dive deeper ☺

Dtrace oneliner (kernel global lock):

```
dtrace -n 'pid$target::kgl*:entry{ @u[probefunc] = count(); } tick-5s {printa ( @u);} tick-15s { exit(0); }' -p PID
```

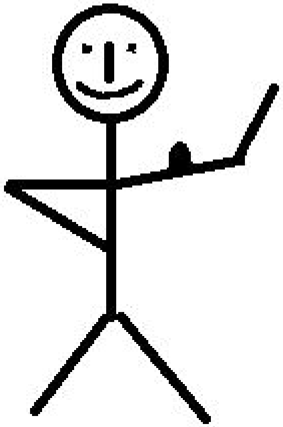
```
kglSetLockSavePoint 3260
kglUnLock 3528
kglLockSetContext 3914
kglLockSetCallbackContext 4070
kglLockUserSession 4342
kglLockCount 4616
kglGetMutex 4640
kglGetSessionUOL 5720
kglReleaseMutex 7324
kglats 39126
```

```
kglSetLockSession 2
kglUnLock 2
kgllkal 2
kglSetLockSavePoint 4
kglGetSessionUOL 5
kglReleaseMutex 5
kglats 20
kglbrk 24
```

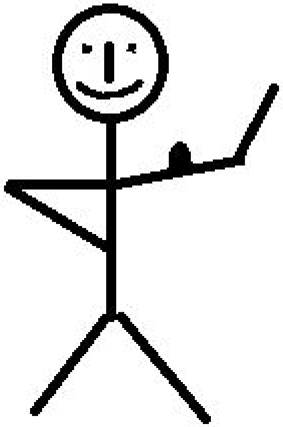
One more: (kernel kursor compile ???):

```
dtrace -n 'pid$target::kks*:entry/pid == $target/ { @u[probefunc] = count(); } tick-5s {printa ( @u);} tick-15s { exit(0); }' -p PID
```

Is our shared pool in a good shape, now?



Is our shared pool in a good shape, now?



- Actually not yet, not completely !
- We have beaten “untyped” binds.

- What’s next?

BIND_LENGTH_UPGRADABLE issue

BIND_LENGTH_UPGRADABLE issue

- Oracle can build a child cursor when it spotted that the particular bind variable length has changed drastically (increased).
- There are following thresholds
  - 32
  - 128
  - 2000
  - 4000
  - …….

Another test

- Let’s prepare another synthetic test
- Create a table with 40 VARCHAR2(4000 BYTE) columns
- Populated with some data
- Start to select with various bind lengths, like
    - “A”, “A”, “A”, ……….. “A”
    - “AA..33….A”, “A”, …, “A”
    - “AA..33 ..A”, “AA…33….A”, ……… “AA…33..A”
    - ………………………………
    - “AA..2001…AA”, …… “AA….2001..AA”

Java code snippet

```
final List<Object[]> batchArgs = new ArrayList<Object[]>();
for (int i = 0; i <= 2; i++) {
    for (int j = 0; j <= 39; j++) {
        Object[] line = new Object[40];
        String[] linestr = new String[40];
        for (int pos = 0; pos <= 39; pos++) {
            if (pos + 1 <= j) {
                //line[pos] = binds[i + 1];
                linestr[pos] = generString(binds[i + 1]);

            } else {
                //line[pos] = binds[i];
                linestr[pos] = generString(binds[i]);
            }
        }
        batchArgs.add(linestr);
    }
}
……………
 simpleJdbcTemplate.query("SELECT /* BindSize40Issue */ * FROM YYTEST " +
                            "WHERE (f1,f2,f3,f4,f5,f6,…,f40) " +
                            "IN ((:f1,:f2,:f3,:f4,:f5,:f6,…,:f40))"
                            , batchArgs.get(j), new RowCallbackHandler() { …
```

Let’s execute it for the 1st time (no cursor in the shared pool)

Bad case – many cursors

parse time cpu	11
parse time elapsed	9
parse count (total)	**130**
parse count (hard)	**176**

Good case - single cursor

?

How it looks like from Oracle side?

Bad case

select child_number, c.**BIND_LENGTH_UPGRADEABLE** from v$sql_shared_cursor c where sql_id='0jdrxy1wavp8n';

CHILD_NUMBER	BIND_LENGTH_UPGRADEABLE
0	N
1	Y
…..	……
120	Y

We see 121 child cursors were created

How it looks like from Oracle side?

Bad case

```
SELECT * FROM (
SELECT c.CHILD_NUMBER, c.NAME,c.DATATYPE_STRING from v$sql_bind_capture c where
sql_id='0jdrxy1wavp8n'
) PIVOT (MAX(DATATYPE_STRING) as MX FOR name IN (':F1',':F2',':F3',':F4',':F5',':F39',':F40') )
ORDER BY 1;
```

CHILD_NUMBER	':F1'_MX	':F2'_MX	':F3'_MX	':F4'_MX	':F5'_MX		':F39'_MX	':F40'_MX
0	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	..	VARCHAR2(32)	VARCHAR2(32)
1	**VARCHAR2(128)**	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	..	VARCHAR2(32)	VARCHAR2(32)
2	**VARCHAR2(128)**	**VARCHAR2(128)**	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	..	VARCHAR2(32)	VARCHAR2(32)
						..		
119	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	..	VARCHAR2(4000)	VARCHAR2(2000)
120	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	..	VARCHAR2(4000)	**VARCHAR2(4000)**

A kind of disclaimer ☺

- Actually Oracle is doing a great job handling those child cursors.
- There is a set of sophisticated algorithms behind scene to have just a few (typically one) such cursors active (in use).
- I even had not been able to construct a representative test case for the issue. (to show a performance problem, not to create a couple of hundreds cursors, which is easy ☺)
- However under high load with a severe concurrency – hundreds sessions execute almost the same cursor you still can get into trouble.

Useful links
https://dioncho.wordpress.com/tag/session-cursor-caching/

Lets change our test a bit

- Table with 40 VARCHAR2(4000 BYTE) columns again
- Populated with some data
- Start to select with various bind lengths, but **in opposite order**
  - “AA$\underbrace{\text{..2001…}}$AA”, …… “AA$\underbrace{\text{…2001..}}$AA”
  - ………………………………
  - “AA..33 ..A”, “AA….A”, ……… “AA…33..A”
  - “AA..33….A”, “A”, ……… “A”
  - “A”, “A”, “A”, ……….. “A”

Java code snippet – ”good case”

```
final List<Object[]> batchArgs = new ArrayList<Object[]>();
for (int i = 0; i <= 2; i++) {
    for (int j = 0; j <= 39; j++) {
        Object[] line = new Object[40];
        String[] linestr = new String[40];
        for (int pos = 0; pos <= 39; pos++) {
            if (pos + 1 <= j) {
                //line[pos] = binds[i + 1];
                linestr[pos] = generString(binds[i + 1]);

            } else {
                //line[pos] = binds[i];
                linestr[pos] = generString(binds[i]);
            }
        }
        batchArgs.add(linestr);
    }
}
 Collections.reverse(batchArgs);
……………
 simpleJdbcTemplate.query("SELECT /* BindSize40Issue */ * FROM YYTEST " +
                            "WHERE (f1,f2,f3,f4,f5,f6,…,f40) " +
                            "IN ((:f1,:f2,:f3,:f4,:f5,:f6,…,:f40))"
                            , batchArgs.get(j), new RowCallbackHandler() { …
```

We just reverse the order of our binds, largest goes 1st !

How it looks like from Oracle side?

Good case

select child_number, c.**BIND_LENGTH_UPGRADEABLE** from v$sql_shared_cursor c where sql_id='0jdrxy1wavp8n';

CHILD_NUMBER	BIND_LENGTH_UPGRADEABLE
0	N

We see a single child cursor was created

Let's execute it for the 1st time (no cursor in the shared pool)

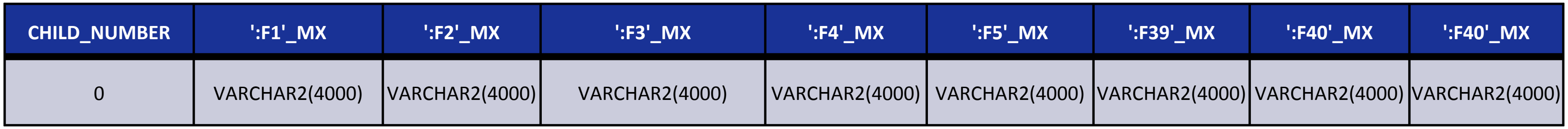
Bad case – many cursors	
parse time cpu	14
parse time elapsed	17
parse count (total)	**408**
parse count (hard)	**124**

Good case - single cursor	
parse time cpu	7
parse time elapsed	3
parse count (total)	408
parse count (hard)	**4**

How it looks like from Oracle side?

Good case

```
SELECT * FROM (
SELECT c.CHILD_NUMBER, c.NAME,c.DATATYPE_STRING from v$sql_bind_capture c where sql_id='0jdrxy1wavp8n'
) PIVOT (MAX(DATATYPE_STRING) as MX FOR name IN (':F1',':F2',':F3',':F4',':F5',':F39',':F40') )
ORDER BY 1;
```



CHILD_NUMBER	':F1'_MX	':F2'_MX	':F3'_MX	':F4'_MX	':F5'_MX	':F39'_MX	':F40'_MX	':F40'_MX
0	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)	VARCHAR2(4000)

Let’s get back to our disclaimer

- Actually Oracle is doing a great job handling those child cursors.
- There is a set of sophisticated algorithms behind scene to have just a few (typically one) such cursors active (in use).

Smart algorithm for “bind length upgradable”

```
-- let’s prepare a test case
-- in SQLPlus this time:

CREATE TABLE BIND_LEN_TST (
 F1 VARCHAR2(512),
 F2 VARCHAR2(512),
 F3 VARCHAR2(512),
 F4 VARCHAR2(512),
 F5 VARCHAR2(512)
);
```

Prepare initial binds

```
VARIABLE F1 VARCHAR2(10)
VARIABLE F2 VARCHAR2(10)
VARIABLE F3 VARCHAR2(10)
VARIABLE F4 VARCHAR2(10)
VARIABLE F5 VARCHAR2(10)

EXEC :F1:= 'AAAA'
EXEC :F2:= 'AAAA'
EXEC :F3:= 'AAAA'
EXEC :F4:= 'AAAA'
EXEC :F5:= 'AAAA'
```

Cleanup and 1st execution

```
EXEC SYS.DBMS_SHARED_POOL.PURGE ('00000006452E2320,1192934567', 'C');

SELECT * FROM BIND_LEN_TST WHERE (F1,F2,F3,F4,F5)
        IN ((:F1,:F2,:F3,:F4,:F5));
```

Result of 1st execution

```
SELECT * from
 (SELECT c.CHILD_NUMBER, c.NAME,c.DATATYPE_STRING from v$sql_bind_capture c
where sql_id='1bx4mmj3jpg57' )
      PIVOT (MAX(DATATYPE_STRING) as MX FOR name IN (':F1'as f1,':F2' as f2,':F3' as
f3,':F4' as f4,':F5' as f5) ) order by 1;
```

CHILD_ NUMBER	F1_MX	F2_MX	F3_MX	F4_MX	F5_MX
0	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)

2nd execution
redefine F1 bind

```
VARIABLE F1 VARCHAR2(20)
EXEC :F1 := 'AAAA'

PAUSE 1st

SELECT * FROM BIND_LEN_TST WHERE (F1,F2,F3,F4,F5)
        IN ((:F1,:F2,:F3,:F4,:F5));
```

We change length

3rd execution
get back F1 bind and redefine F2 bind

```
VARIABLE F1 VARCHAR2(10)
VARIABLE F2 VARCHAR2(20)
EXEC :F1 := 'AAAA'
EXEC :F2 := 'AAAA'

PAUSE 2

SELECT * FROM BIND_LEN_TST WHERE (F1,F2,F3,F4,F5)
       IN ((:F1,:F2,:F3,:F4,:F5));
```

Get back length

We change length

Result of 2 & 3 execution

```
SELECT * from
 (SELECT c.CHILD_NUMBER, c.NAME,c.DATATYPE_STRING from v$sql_bind_capture c
where sql_id='1bx4mmj3jpg57' )
      PIVOT (MAX(DATATYPE_STRING) as MX FOR name IN (':F1'as f1,':F2' as f2,':F3' as
f3,':F4' as f4,':F5' as f5) ) order by 1;
```

CHILD_ NUMBER	F1_MX	F2_MX	F3_MX	F4_MX	F5_MX
0	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)
0	**VARCHAR2(128)**	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)
0	**VARCHAR2(128)**	**VARCHAR2(128)**	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)

Result of 4 & 5 execution

```
SELECT * from
 (SELECT c.CHILD_NUMBER, c.NAME,c.DATATYPE_STRING from v$sql_bind_capture c
where sql_id='1bx4mmj3jpg57' )
      PIVOT (MAX(DATATYPE_STRING) as MX FOR name IN (':F1'as f1,':F2' as f2,':F3' as
f3,':F4' as f4,':F5' as f5) ) order by 1;
```

CHILD_ NUMBER	F1_MX	F2_MX	F3_MX	F4_MX	F5_MX
0	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)
1	**VARCHAR2(128)**	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)
2	**VARCHAR2(128)**	**VARCHAR2(128)**	VARCHAR2(32)	VARCHAR2(32)	VARCHAR2(32)
3	**VARCHAR2(128)**	**VARCHAR2(128)**	**VARCHAR2(128)**	VARCHAR2(32)	VARCHAR2(32)
4	**VARCHAR2(128)**	**VARCHAR2(128)**	**VARCHAR2(128)**	**VARCHAR2(128)**	VARCHAR2(32)

Smart Algorithm

Let’s see what we actually discovers:

Actual bind length

	1	2	3	4	5
32					
128	🟩				

	1	2	3	4	5
32					
128		🟩			

	1	2	3	4	5
32					
128			🟩		

	1	2	3	4	5
32					
128				🟩	

	1	2	3	4	5
32					
128					🟩

Cursor created

	1	2	3	4	5
32					
128	🟩				

	1	2	3	4	5
32					
128	🟩	🟩			

	1	2	3	4	5
32					
128	🟩	🟩	🟩		

	1	2	3	4	5
32					
128	🟩	🟩	🟩	🟩	

	1	2	3	4	5
32					
128	🟩	🟩	🟩	🟩	🟩

Additional cursor won't be created for new combinations: N vs N!

Actual bind length

	1	2	3	4	5
32					
128		🟩			

	1	2	3	4	5
32					
128	🟩		🟩		

	1	2	3	4	5
32					
128	🟩	🟩		🟩	

	1	2	3	4	5
32					
128	🟩		🟩		🟩

Cursor created

	1	2	3	4	5
32					
128	🟩	🟩			

	1	2	3	4	5
32					
128	🟩	🟩	🟩		

	1	2	3	4	5
32					
128	🟩	🟩	🟩	🟩	

	1	2	3	4	5
32					
128	🟩	🟩	🟩	🟩	🟩

HARD parse is not always very hard ☺

parse count (total) – number of *‘parse calls’*. If the statement has never been seen before, this will result in parsing and optimisation; if the statement has been seen before this will involve a search of the ***library cache*** (which is how you discover it’s been seen before) and may result in *‘cursor authentication’*; if the statement has been seen and authenticated before and has a reference in the ***session cursor cache*** then the *‘parse call’* may do virtually nothing – because there isn’t even a need to search the ***library cache*** for the statement because it has been implicitly held.

parse count (hard) – number of optimisations that took place. Optimisation may be a consequence of a *‘parse call’* or an *‘execute call’*. Even a statement that is being ***held*** can have it’s plan invalidated or simply flushed from the ***library cache***, which leads to optimisation on the next execution.

Useful links
https://jonathanlewis.wordpress.com/2007/07/03/parse-calls/

Really hard parse

Really hard parse (literals)	11.2.0.4 latest patches
parse time cpu :::: 12	parse time cpu :::: 1
parse time elapsed :::: 9	parse time elapsed :::: 0
parse count (total) :::: 40	parse count (total) :::: 2
parse count (hard) :::: 40	parse count (hard) :::: 40
parse count (failures) :::: 0	parse count (failures) :::: 0
parse count (describe) :::: 0	parse count (describe) :::: 0
execute count :::: 200	execute count :::: 200

Auto-commit issue

Auto-commit issue

“Auto-commit = true“ is specified by JDBC standard

JDBC 3.0 specification:
10.1.1

The default is for **auto-commit** mode to be **enabled** when the Connection object is created.

JDBC 4.2 specification:

10.1.1 - the same statement.

The default is for **auto-commit** mode to be **enabled** when the Connection object is created.

Let’s get back to our previous example

Good case

SQL*Net roundtrips to/from client	3
parse count (total)	17
parse count (hard)	0
user calls	6
user commits	**1**

Bad case

SQL*Net roundtrips to/from client	**553**
parse count (total)	**566**
parse count (hard)	1
user calls	**555**
user commits	**550**

When auto-commit set ON - Oracle driver commits each piece.

Bad case – auto-commit is on

✓ It can not only lead to performance issue

but also semantic change!!!

✓ JDBC batches are usually executed as a single transaction, I’ve checked it with various driver from 10.x to 11.2 but in this case that’s wrong.

✓ Extract from official Oracle’s recommendations:
Disable auto-commit mode if you use **either** update **batching model**. In case an error occurs while you are processing a batch, this provides you the option of committing or rolling back the operations that ran successfully prior to the error.

Either means here Oracle Implementation or Standard JDBC batching.

Let’s change our previous example a bit
(we just wrapped our call into TransactionTemplate)

```
SimpleJdbcTemplate simpleJdbcTemplate = new SimpleJdbcTemplate(dataSource);

List<Object[]> batchArgs = new ArrayList<Object[]>();

for (int i = 1; i <= 1000; i++) {
    Map<String, Object> map = new HashMap<String, Object>();
    Object[] item = new Object[8];
    item[0] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[1] = getRandomInt() % 10 == 5 ? null : getRandomString();
    item[2] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[3] = getRandomInt() % 10 == 5 ? null : getRandomString();
    item[4] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[5] = getRandomInt() % 10 == 5 ? null : getRandomString();
    item[6] = getRandomInt() % 10 == 5 ? null : getRandomNumber();
    item[7] = getRandomInt() % 10 == 5 ? null : getRandomString();
    batchArgs.add(item);
}
 transactionTemplate.execute(new TransactionCallbackWithoutResult() {
   @Override
  protected void doInTransactionWithoutResult(TransactionStatus status) {
    SimpleJdbcTemplate simpleJdbcTemplate = new SimpleJdbcTemplate(dataSource);

    long startTime = System.currentTimeMillis();
    simpleJdbcTemplate.batchUpdate("INSERT /* BatchIssueSpring */ INTO YYYTEST
(f1,f2,f3,f4,f5,f6,f7,f8) VALUES (:f1,:f2,:f3,:f4,:f5,:f6,:f7,:f8)", batchArgs);
    System.out.println("time " + ((double) (System.currentTimeMillis() - startTime)) / 1000);
```

We wrap our JDBC code with TransactionTemplate

As a result of the change.
We get rid of 1 of 3 issues

Good case

Doesn’t matter

Bad case

SQL*Net roundtrips to/from client	**553**
parse count (total)	**566**
parse count (hard)	**0**
user calls	**562**
user commits	**1**

Commit rate get back to normal

Always set auto-commit off when works with JDBC batches!!!

- TransactionTemplate code (or @Transaction annotation) explicitly sets auto-commit off.

Disclaimer

This is not an offer to provide any services. This material is for information and illustrative purposes only and is not intended, nor should it be distributed, for advertising purposes, nor is it intended for publication or broadcast. Any third party analysis does not constitute any endorsement or recommendation. Opinions expressed herein are current opinions as of the date appearing in this material only and are subject to change without notice This information is provided with the understanding that with respect to the material provided herein, that you will make your own independent decision with respect to any course of action in connection herewith and as to whether such course of action is appropriate or proper based on your own judgment, and that you are capable of understanding and assessing the merits of a course of action. "Deutsche Bank TechCentre" LLC shall not have any liability for any damages of any kind whatsoever relating to this material.

Q&A

Difference between Oracle versions

11.2.0.3	11.2.0.4 latest patches
parse time cpu :::: 1	parse time cpu :::: 0
parse time elapsed :::: 2	parse time elapsed :::: 0
parse count (total) :::: 2	**parse count (total) :::: 42**
parse count (hard) :::: 40	**parse count (hard) :::: 40**
parse count (failures) :::: 0	parse count (failures) :::: 0
parse count (describe) :::: 0	parse count (describe) :::: 0
execute count :::: 200	execute count :::: 240