

PL/SQL vs. JavaScript in the Oracle Database 23c

Philipp Salvisberg
23rd April 2024

Philipp Salvisberg

Data Engineering Principal

- Database Centric Development
- Model Driven Software Development
- Open-Source Development

philipp.salvisberg@accenture.com

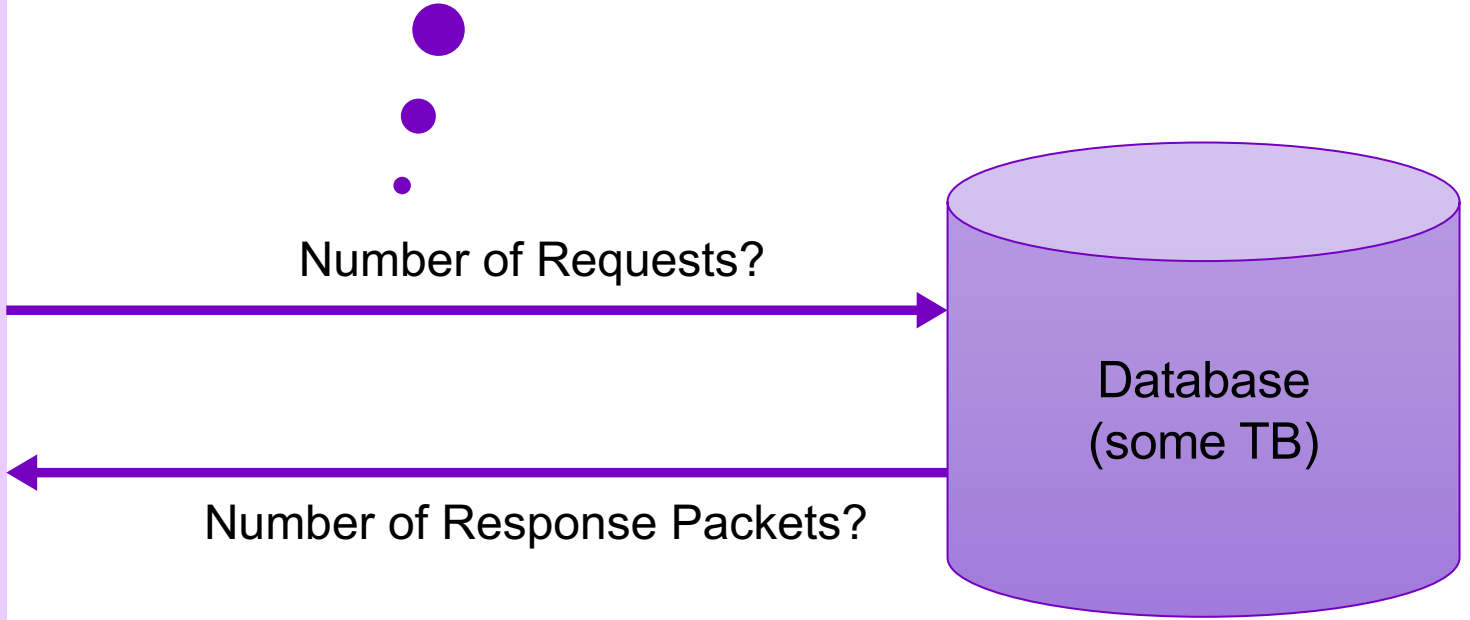
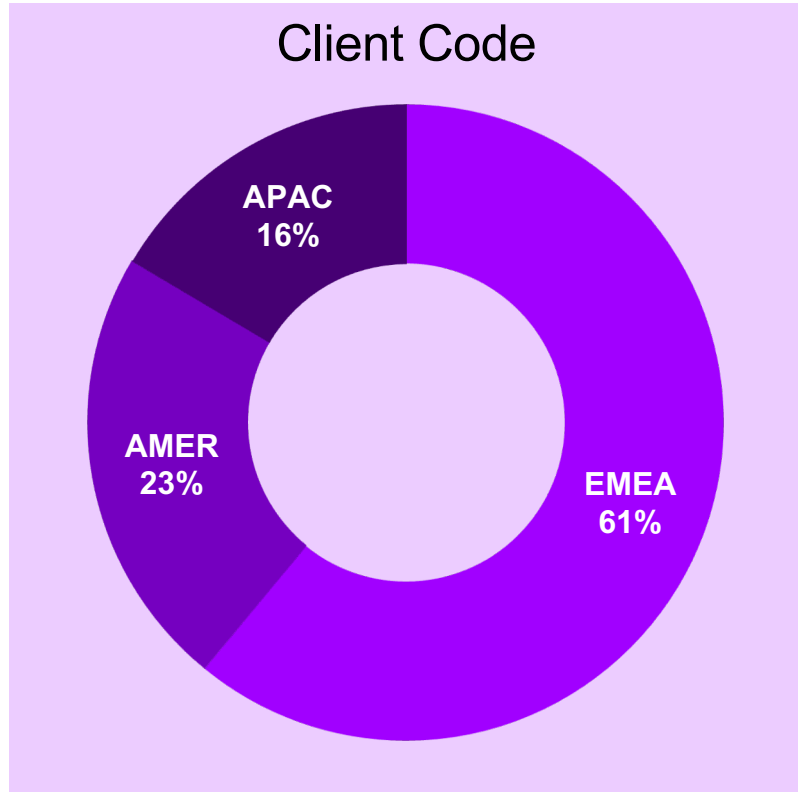
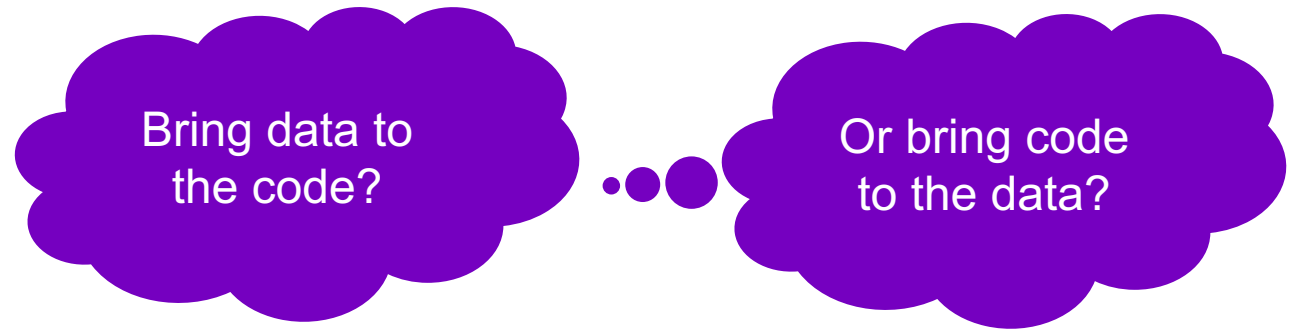
<https://www.salvis.com/blog>

<https://github.com/PhilippSalvisberg/js23c/tree/main>



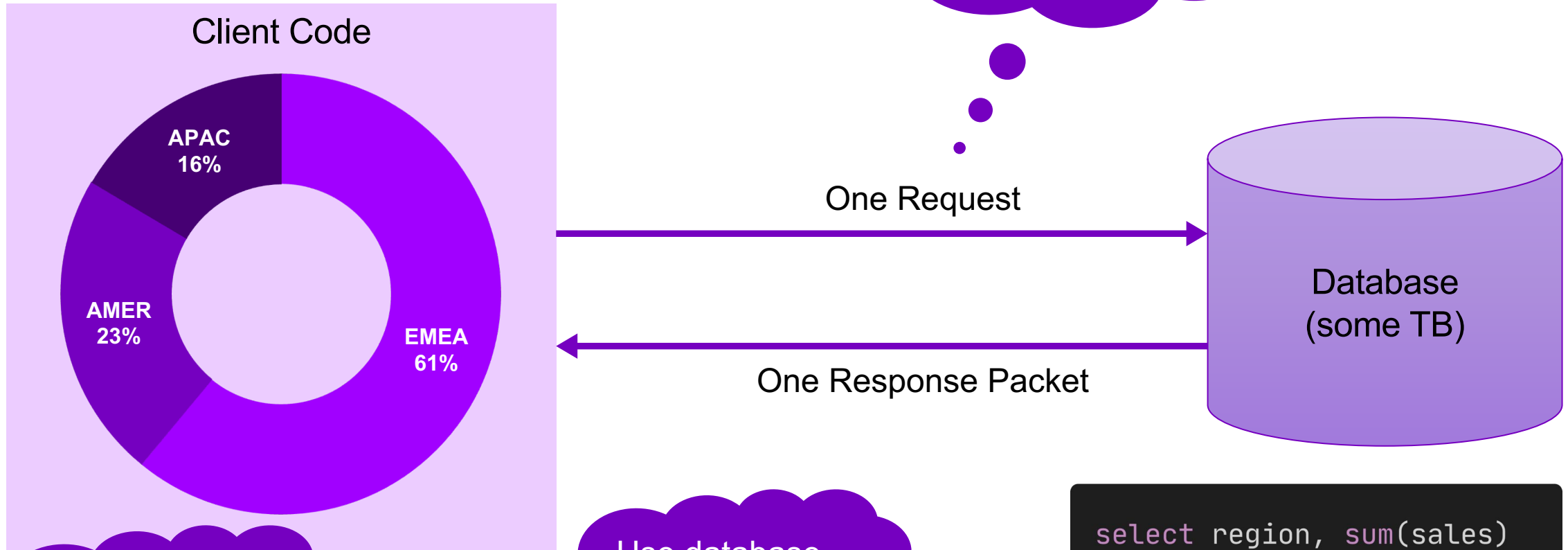
Why Do We Need Code in the Database?

Processing Data



Bring Code to the Data

Minimize communication for good performance



Make the DB responsible for the data quality

Use database as processing engine

```
select region, sum(sales)
from ...
group by region
```

Programming Languages in the Oracle Database

C (Version 8.0)

Shared library must be installed on the database server's file system

```
1 create or replace procedure utl_xml_parse_query(  
2   in_current_userid in      number,  
3   in_schema_name   in      varchar2,  
4   in_query         in      clob,  
5   io_result        in out nocopy clob  
6 ) is  
7   language c  
8   library sys.utl_xml_lib      -- static lib, part of the oracle binary  
9   name "kuxParseQuery"        -- name of the C function in the lib  
10  with context parameters (  
11    context,                  -- connection context, allows callbacks  
12    in_current_userid ocinumber, -- content  
13    in_current_userid indicator, -- null indicator  
14    in_schema_name ocistring,   -- content  
15    in_schema_name indicator,  -- null indicator  
16    in_query ociloblocator,    -- content  
17    in_query indicator,        -- null indicator  
18    io_result ociloblocator,   -- content  
19    io_result indicator        -- null indicator  
20 );  
21 /
```

PL/SQL (Version 7.0)

```
1 create or replace procedure increase_salary_plsql(  
2   in_deptno      in number,  
3   in_by_percent  in number  
4 ) is  
5 begin  
6   update emp  
7     set sal = sal + sal * in_by_percent / 100  
8     where deptno = in_deptno;  
9 end increase_salary_plsql;  
10 /
```

Usages in the
data dictionary

Compile errors
& no SQLi

Static
SQL

PL/SQL – Without Data Processing (DML)?

```
1 create or replace function to_epoch_plsql(in_ts in timestamp) return number is
2   co_epoch_date constant timestamp with time zone := timestamp '1970-01-01 00:00:00 UTC';
3   l_interval      interval day(9) to second (3);
4 begin
5   l_interval := in_ts - co_epoch_date;
6   return 1000 * (extract(second from l_interval)
7     + extract(minute from l_interval) * 60
8     + extract(hour from l_interval) * 60 * 60
9     + extract(day from l_interval) * 60 * 60 * 24);
10 end;
11 /
```

Is it necessary
that we implement
that ourselves?

Java (Version 8.1)



```
1 create or replace and compile java source named "Util" as
2 public class Util {
3     public static long toEpoch(java.sql.Timestamp ts) {
4         return ts.getTime();
5     }
6 }
7 /
```

PL/SQL wrapper for
accessibility and
data type conversions



```
1 create or replace function to_epoch_java(in_ts in timestamp)
2     return number is language java name
3     'Util.toEpoch(java.sql.Timestamp) return java.lang.long';
4 /
```

Java – Using 3rd Party Libraries

```
1 loadjava -thin \  
2   -user myuser/mypassword@localhost:1521:orcl \  
3   -genmissing \  
4   -resolve \  
5   -resolver "((* MYUSER) ((* PUBLIC) ((* -)))" \  
6   -verbose \  
7   -stdout \  
8   mysql-connector-java-5.0.8-bin.jar
```

Must be compatible with
the OJVM in the database

1.8.0_371 in 19.19
11.0.20 in 23.3

Every class/resource
becomes an object in the
Oracle data dictionary

Java – Permissions (Network, Filesystem, ...)

```
1 declare
2   co_user constant all_users.username%type := 'DEMO';
3 begin
4   dbms_java.grant_permission(co_user,
5     'SYS:java.net.SocketPermission', '*:1024-65535', 'connect, resolve');
6   dbms_java.grant_permission(co_user,
7     'SYS:java.io.FilePermission', '/tmp', 'read, write');
8   dbms_java.grant_permission(co_user,
9     'SYS:java.lang.RuntimePermission', 'getClassLoader', '');
10  dbms_java.grant_permission(co_user,
11    'SYS:java.lang.RuntimePermission', 'setContextClassLoader', '');
12 end;
13 /
```

Dynamic JavaScript (Version 21)

```
1 create or replace function to_epoch_djs(in_ts in timestamp) return number is
2   co_js      constant clob := q'~
3     const bindings = require("mle-js-bindings");
4     const ts = bindings.importValue("ts");
5     bindings.exportValue("millis", ts.valueOf());
6   ~';
7   l_ctx      dbms_mle.context_handle_t;
8   l_millis number;
9 begin
10  l_ctx := dbms_mle.create_context();
11  dbms_mle.export_to_mle(l_ctx, 'ts', in_ts);
12  dbms_mle.eval(l_ctx, 'JAVASCRIPT', co_js);
13  dbms_mle.import_from_mle(l_ctx, 'millis', l_millis);
14  dbms_mle.drop_context(l_ctx);
15  return l_millis;
16 end to_epoch_djs;
17 /
```

mle-js-oracle-db,
mle-js-plsqltypes,
mle-js-fetch (23c),
mle-encode-base64 (23c)

Map input and output
types dynamically

Static JavaScript (Version 23)

```
1 create or replace mle module util_mod language javascript as
2 export function toEpoch(ts) {
3     return ts.valueOf();
4 }
5 /
```

PL/SQL wrapper for
accessibility and
optional data type conversions

```
1 create or replace function to_epoch_js(in_ts in timestamp)
2     return number is
3     mle module util_mod
4     signature 'toEpoch(Date)'; -- conversion to OracleTimestamp is also possible
5 /
```

Inline MLE Call Specification

```
1 create or replace mle module util_mod language javascript as
2 export function toEpoch(ts) {
3   return ts.valueOf();
4 }
5 /
```

```
1 create or replace function to_epoch_js(in_ts in timestamp)
2   return number is
3   mle module util_mod
4   signature 'toEpoch(Date)', conversion to OracleTimestamp is also possible
5 /
```

```
1 create or replace function to_epoch_js2("in_ts" in timestamp)
2   return number is
3   mle language javascript q'[return in_ts.valueOf();]';
4 /
```

JavaScript function

SQL in JavaScript

```
1 create or replace mle module increase_salary_mod language javascript as
2 export function increase_salary(deptno, by_percent) {
3   session.execute(`
4     update emp
5       set sal = sal + sal * :by_percent / 100
6       where deptno = :deptno`, [by_percent, deptno]);
7 }
8 /
```

No need to import
mle-js-oracledb

Dynamic SQL
using ES6
template literals

Runtime errors
& SQLi risk

No usages in the
data dictionary

JavaScript – 3rd Party Libraries

Requires SQLcl or SQL Developer

```
1 set define off
2 script
3 var url = new java.net.URL("https://esm.run/validator@13.11.0");
4 var content = new java.lang.String(url.openStream().readAllBytes(),
5     java.nio.charset.StandardCharsets.UTF_8);
6 var script = "create or replace file module validator_mod "
7     + "language javascript as " + "\n"
8     + content + "\n"
9     + "/" + "\n";
10 sqlcl.setScript(script);
11 sqlcl.run();
12 /
```

Disconnected from npm

Single object in the Oracle data dictionary

```
1 mle install validator_mod https://esm.run/validator@13.11.0 13.11.0
```

Custom SQLcl Command



JavaScript – MLE Module Wrapper

```
1 create or replace package validator_api is
2   function is_email(
3     in_email in varchar2
4   ) return boolean as mle module validator_mod signature 'default.isEmail(string)';
5
6   function is_email(
7     in_email in varchar2,
8     in_options in json
9   ) return boolean as mle module validator_mod signature 'default.isEmail(string, any)';
10
11  function is_email_djs(
12    in_email in varchar2,
13    in_options in json default null
14  ) return boolean;
15 end validator_api;
16 /
```

Does it make sense to expose in_options?

Default values for parameters are not supported

JavaScript wrapper in package body

Validator Documentation for isEmail “options”

Default options:

```
{
  allow_display_name: false,
  allow_underscores: false,
  require_display_name: false,
  allow_utf8_local_part: true,
  require_tld: true,
  allow_ip_domain: false,
  domain_specific_validation: false,
  blacklisted_chars: '',
  ignore_max_length: false,
  host_blacklist: [],
  host_whitelist: []
}
```

If **allow_display_name** is set to true, the validator will also match Display Name <email-address>.

If **require_display_name** is set to true, the validator will reject strings without the format Display Name <email-address>.

If **allow_utf8_local_part** is set to false, the validator will not allow any non-English UTF8 character in email address' local part.

If **require_tld** is set to false, email addresses without a TLD in their domain will also be matched.

If **allow_ip_domain** is set to true, the validator will allow IP addresses in the host part.

If **domain_specific_validation** is true, some additional validation will be enabled, e.g. disallowing certain syntactically valid email addresses that are rejected by Gmail.

If **blacklisted_chars** receives a string, then the validator will reject emails that include any of the characters in the string, in the name part.

If **ignore_max_length** is set to true, the validator will not check for the standard max length of an email (254).

If **host_blacklist** is set to an array of strings and the part of the email after the @ symbol matches one of the strings defined in it, the validation fails.

If **host_whitelist** is set to an array of strings and the part of the email after the @ symbol matches none of the strings defined in it, the validation fails



JavaScript – Importing MLE Modules

```
1 create or replace package body validator_api is
2   function is_email_djs(
3     in_email   in varchar2,
4     in_options in json default null
5   ) return boolean is
6     co_js      constant clob := q'~
7       (async () => {
8         const bindings = await import("mle-js-bindings");
9         const email = bindings.importValue("email");
10        const options = bindings.importValue("options");
11        const validator = await import("validator");
12        const result = validator.default.isEmail(email, options);
13        bindings.exportValue("result", result);
14      })();
15     ~';
16     l_ctx      dbms_mle.context_handle_t;
17     l_options  json;
18     l_result  boolean;
19   begin
20     l_options := coalesce(in_options, JSON('{}'));
21     l_ctx := dbms_mle.create_context(environment => 'DEMO_ENV');
22     dbms_mle.export_to_mle(l_ctx, 'email', in_email);
23     dbms_mle.export_to_mle(l_ctx, 'options', l_options);
24     dbms_mle.eval(l_ctx, 'JAVASCRIPT', co_js);
25     dbms_mle.import_from_mle(l_ctx, 'result', l_result);
26     dbms_mle.drop_context(l_ctx);
27     return l_result;
28   end is_email_djs;
29 end validator_api;
30 /
```

Import requires
async/await interface
in dynamically executed
JavaScript

Available imports
& compile options

JavaScript – MLE Environments

```
1 create or replace mle env demo_env
2   imports(
3     'create_temp_table' module create_temp_table_mod,
4     'increase_salary'   module increase_salary_mod,
5     'sql-assert'        module sql_assert_mod,
6     'validator'         module validator_mod,
7     'util'              module util_mod
8   )
9   language options 'js.strict=true, js.console=false, js.polyglot-builtin=true';
```

Modules that
can be imported
from JavaScript

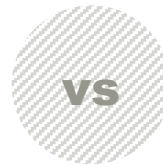
Can handle
different module
versions

Compile options

Languages in the Oracle Database 23c

Connection is Enough

- Primarily
 - SQL
 - PL/SQL (anonymous blocks, with_clause)
 - JavaScript (DBMS_MLE)
- Secondarily
 - SQL/PGQ
 - SQL/JSON
 - SQL/XML
 - XPath
 - XSLT
 - XQuery



Code in the DB is Required

- C
 - Requires access to the DB server to install shared libraries
- PL/SQL
 - Procedures, Functions, Packages, Types, Triggers, Views
 - Static SQL, compile-time dependencies
- Java
 - Requires loadjava to install JARs
 - Simple Java source via SQL
- JavaScript
 - Requires scripting to install libraries
 - Simple JavaScript source via SQL

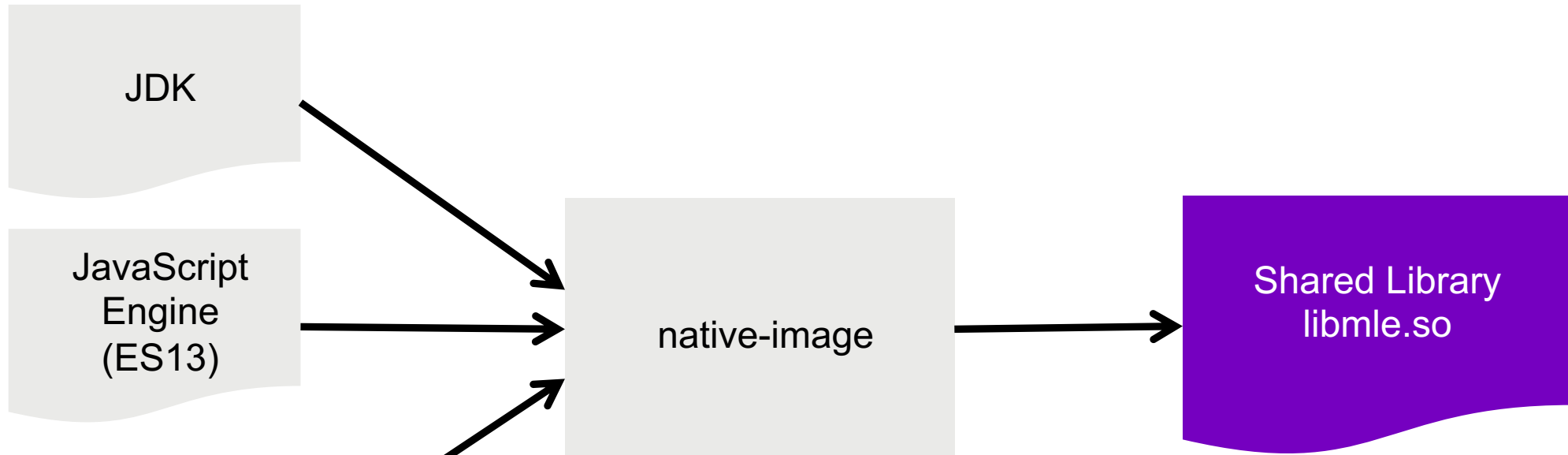
How Many JVMs are in the Oracle Database?

Java – Full JVM as Part of the DB (Option)

```
$ pwd
/u01/app/oracle/product/19.0.0/dbhome/javavm
$ du -h | sort -k2
437M      .
12K       ./admin
56K       ./doc
612K     ./install
32K       ./install/sbs
430M     ./jdk
430M     ./jdk/jdk8
397M     ./jdk/jdk8/admin
34M       ./jdk/jdk8/lib
168K     ./jdk/jdk8/lib/security
4.0M     ./lib
340K     ./lib/cmm
2.0M     ./lib/fonts
8.0K     ./lib/security
2.6M     ./ojvmwcu
8.0K     ./ojvmwcu/bin
36K      ./ojvmwcu/install
2.5M     ./ojvmwcu/lib
```

```
$ pwd
/opt/oracle/product/23c/dbhomeFree/javavm
$ du -h | sort -k2
229M     .
20K      ./admin
108K     ./conf
24K      ./conf/management
84K      ./conf/security
44K      ./doc
700K     ./install
28K      ./install/sbs
220M     ./jdk
220M     ./jdk/jdk11
185M     ./jdk/jdk11/admin
36M      ./jdk/jdk11/lib
120K     ./jdk/jdk11/lib/security
4.0M     ./lib
336K     ./lib/cmm
2.0M     ./lib/fonts
16K      ./lib/security
3.5M     ./ojvmwcu
4.0K     ./ojvmwcu/bin
40K      ./ojvmwcu/install
3.5M     ./ojvmwcu/lib
```


MLE/JS – Native Image as Part of the DB



```
1 $ pwd
2 /opt/oracle/product/23c/dbhomeFree/lib
3 $ ls -lh libmle.so
4 -rw-r----- 1 oracle oinstall 195M Sep  1 19:00 libmle.so
```

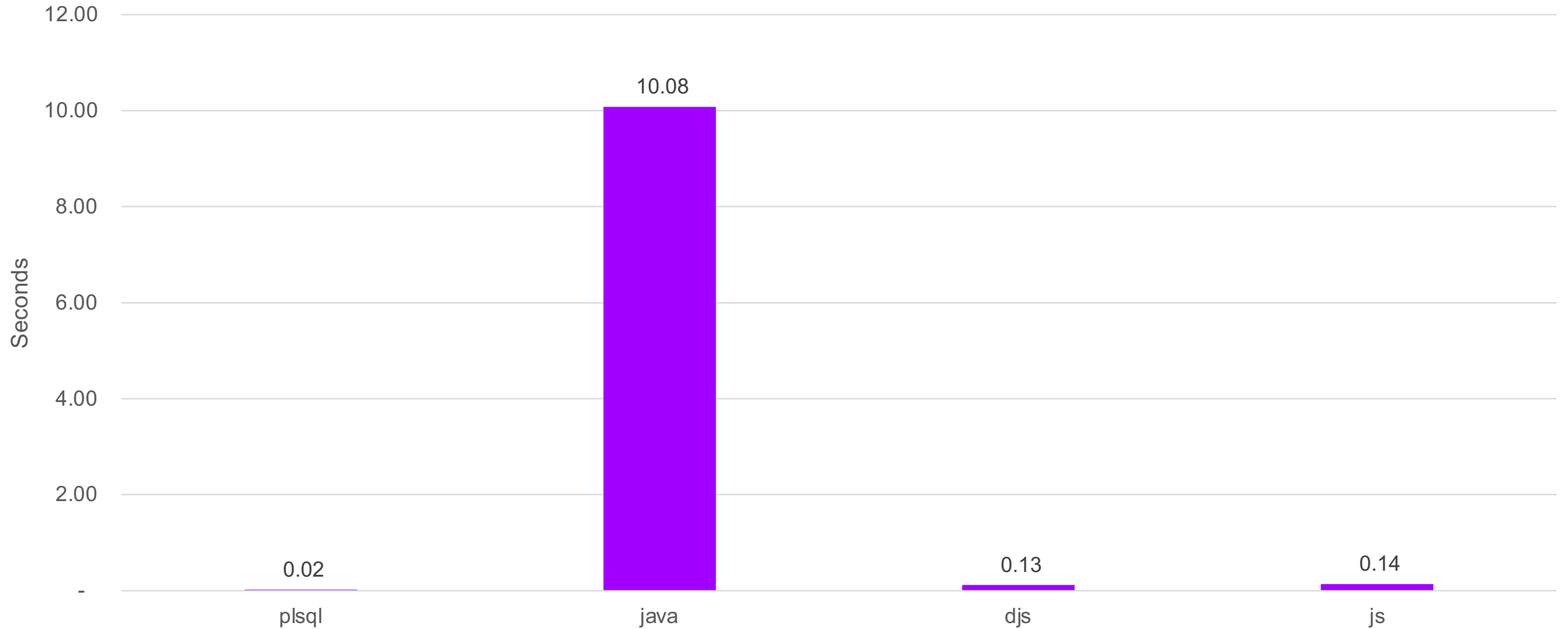


Comparing Performance & Resource Usage

**Best of 3
Attempts**

>

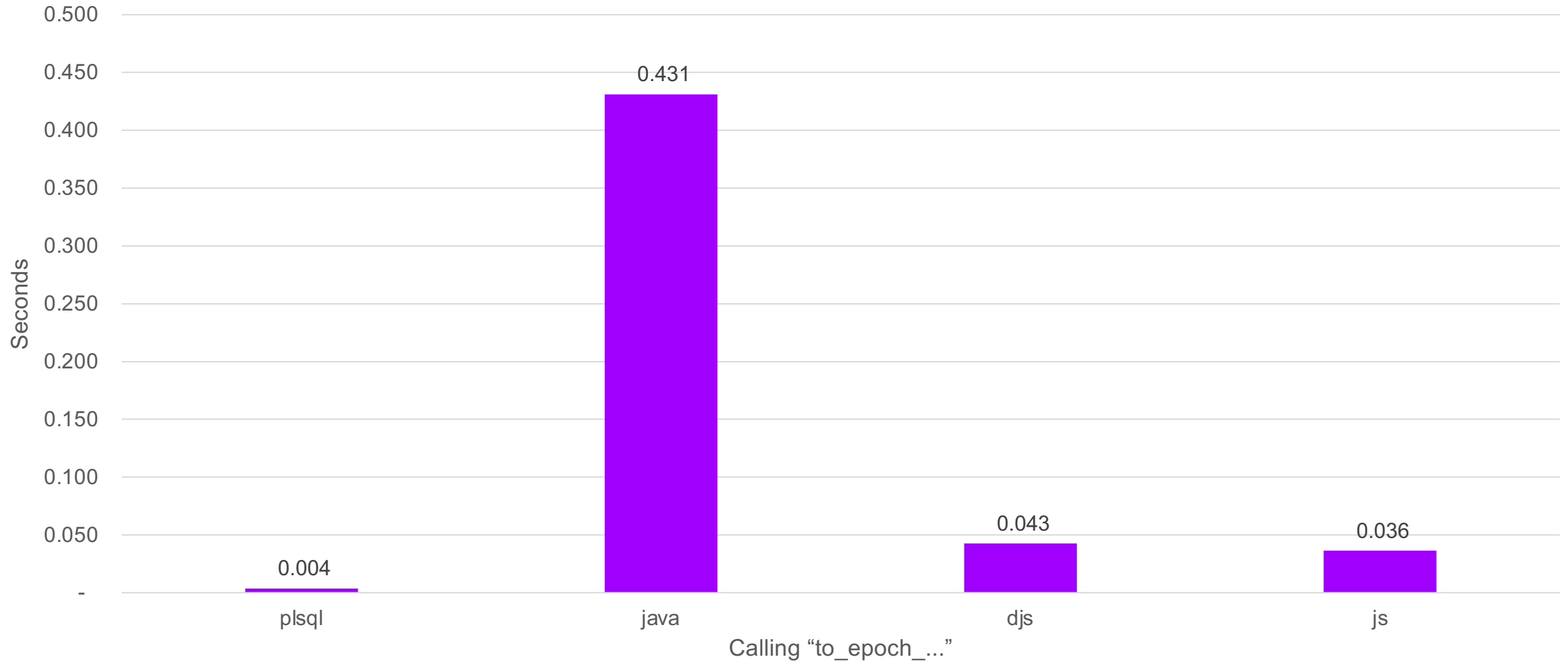
Runtime of First Call after DB Restart



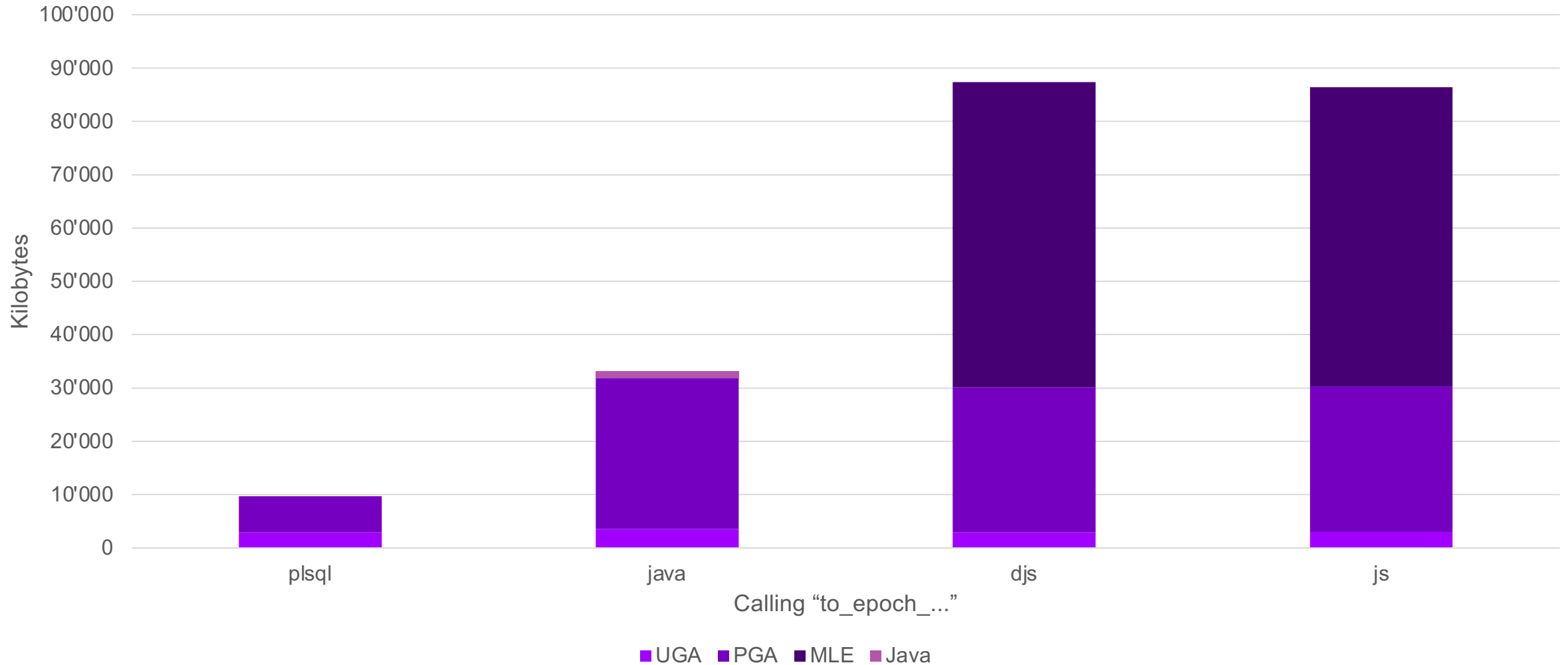
Calling "to_epoch_..."



Runtime of First Call in New DB Session



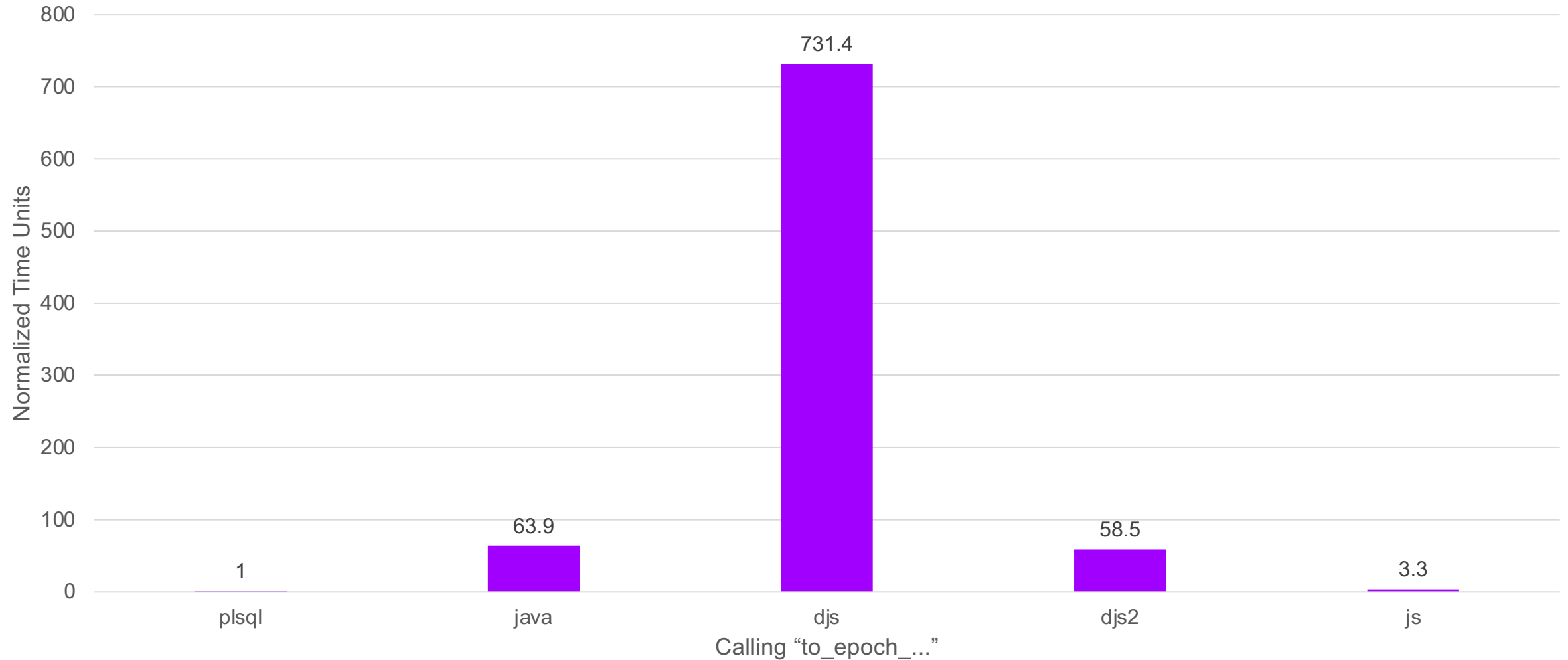
Max. Memory Usage After Single Call



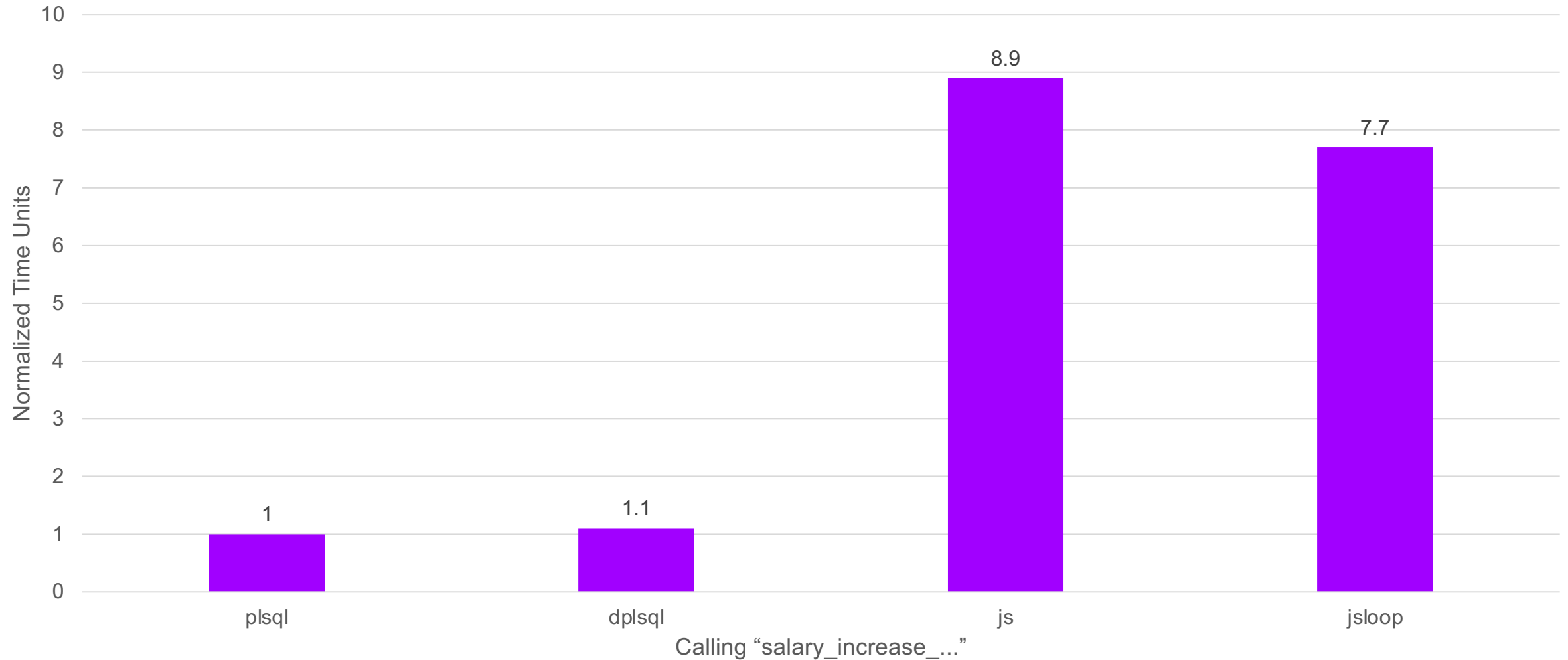
Comparing Apples with Pears?



Runtime of 100'000 Calls of to_epoch



Runtime of 100'000 Calls of salary_increase



Good Security Practices & Default Exceptions

Binds and Assertions

```
1 create or replace mle module
2   create_temp_table_mod language javascript as
3
4 export function createTempTable(tableName) {
5   const result = session.execute(
6     `select dbms_assert.simple_sql_name(
7       :tableName
8     ) as tab`,
9     [tableName]
10  );
11
12  session.execute(
13    `create private temporary table
14      ora\${ptt_${result.rows[0].TAB}} (id number)`
15  );
16 }
17 /
```

```
1 create or replace procedure create_temp_table_plsql(
2   in_table_name in varchar2
3 ) is
4   co_tmpl constant varchar2(1000 char) :=
5     'create private temporary table
6       ora$ptt_#valid_table_name# (id number)';
7 begin
8   execute immediate replace(co_tmpl,
9     '#valid_table_name#',
10    dbms_assert.simple_sql_name(in_table_name));
11 end;
12 /
```

Template literals and
JSON integration are great

Assert in JavaScript

```
1 create or replace mle module
2   create_temp_table_mod language javascript as
3
4 import { simpleSqlName } from "sql-assert";
5
6 export function createTempTable(tableName) {
7   session.execute(
8     `create private temporary table
9       ora\${ptt_}${simpleSqlName(tableName)} (id number)`
10  );
11 }
12 /
```

```
1 SQL> exec create_temp_table_js('my-table');
2
3 Error starting at line : 1 in command -
4 BEGIN create_temp_table_js('my-table'); END;
5 Error report -
6 ORA-04161: Error: Invalid SQL name.
7 ORA-04171: at e (DEMO.SQL_ASSERT_MOD:7:463)
8 ORA-06512: at "DEMO.CREATE_TEMP_TABLE_JS", line 1
9 ORA-06512: at line 1
10 04161. 00000 - "%s"
11 *Cause:   A runtime error occurred while evaluating a Multilingual Engine
12           (MLE) code snippet or call specification.
13 *Action:  Fix the MLE language code that causes the runtime error. Use the
14           source location reported in the error message to identify the code
15           that needs to be fixed. Use the DBMS_MLE.get_error_stack or
16           DBMS_MLE.get_ctx_error_stack functions to retrieve the MLE language
17           stack trace for the error.
18
19 More Details :
20 https://docs.oracle.com/error-help/db/ora-04161/
21 https://docs.oracle.com/error-help/db/ora-04171/
```

Good ORA-04161 message,
get missing references via
dbms_mle.get_error_stack...

Get Missing Error Stack

```
1 SQL> exec create_temp_table_js('my-table');
2
3 Error starting at line : 1 in command -
4 BEGIN create_temp_table_js('my-table'); END;
5 Error report -
6 ORA-04161: Error: Invalid SQL name.
7 ORA-04171: at e (DEMO.SQL_ASSERT_MOD:7:463)
8 ORA-06512: at "DEMO.CREATE_TEMP_TABLE_JS", line 1
9 ORA-06512: at line 1
10 04161. 00000 - "%s"
11 *Cause:      A runtime error occurred while evaluating a Multilingual Engine
12              (MLE) code snippet or call specification.
13 *Action:     Fix the MLE language code that causes the runtime error. Use the
14              source location reported in the error message to identify the code
15              that needs to be fixed. Use the DBMS_MLE.get_error_stack or
16              DBMS_MLE.get_ctx_error_stack functions to retrieve the MLE language
17              stack trace for the error.
18
19 More Details :
20 https://docs.oracle.com/error-help/db/ora-04161/
21 https://docs.oracle.com/error-help/db/ora-04171/
```

```
1 set serveroutput on size unlimited
2 declare
3   t_frames dbms_mle.error_frames_t;
4 begin
5   t_frames := dbms_mle.get_error_stack(
6               module_name => 'CREATE_TEMP_TABLE_MOD',
7               env_name    => 'DEMO_ENV'
8               );
9   for i in t_frames.count
10  loop
11     dbms_output.put_line(
12         'ORA-04171: at '
13         || t_frames(i).func
14         || ' ('
15         || t_frames(i).source
16         || ':'
17         || t_frames(i).line
18         || ':'
19         || t_frames(i).col
20         || ')'
21     );
22   end loop;
23 end;
24 /
25
26 ORA-04171: at createTempTable (DEMO.CREATE_TEMP_TABLE_MOD:6:20)
27
28
29 PL/SQL procedure successfully completed.
```



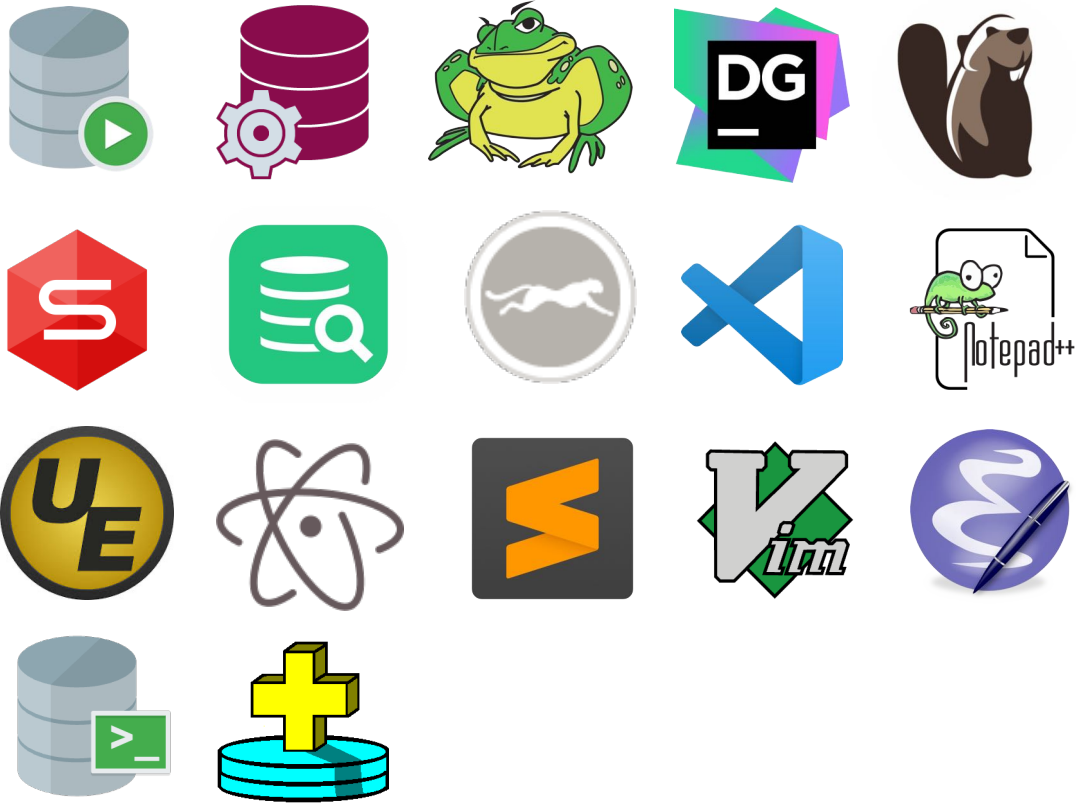
John McEnroe, June 22, 1981

"You cannot be serious!"

Development & Tooling

Development Tools

SQL & PL/SQL



JavaScript



IntelliSense in VS Code

```
sandbox > test > JS demo-validator.js > describe('Validator') callback > describe('isEmail()') callback > it('Philipp Salvisberg <philipp@salvis.com> is valid when allowing display name') callback
You, now | 1 author (You)
1 import assert from 'assert';
2 import validator from 'validator';
3
4 describe('Validator', function () {
5   describe('isEmail()', function () {
6     it('philipp@salvis.com is valid', function () {
7       assert(validator.isEmail('philipp@salvis.com'));
8     });
9     it('Philipp Salvisberg <philipp@salvis.com> is invalid with default options', function () {
10      assert(validator.isEmail('Philipp Salvisberg <philipp@salvis.com>') === false);
11    });
12    it('Philipp Salvisberg <philipp@salvis.com> is valid when allowing display name', function () {
13      assert(validator.isEmail('Philipp Salvisberg <philipp@salvis.com>', {allow_display_name: true, }));
14    });
15  });
16 });
```

(property) IsEmailOptions.allow_ip_domain? ×
n?: boolean | undefined

If allow_ip_domain is set to true, the validator will allow IP addresses in the host part.

@default
false

- allow_ip_domain?
- allow_utf8_local_part?
- blacklisted_chars?
- domain_specific_validation?
- host_blacklist?
- host_whitelist?
- ignore_max_length?
- require_display_name?
- require_tld?
- abc allow_display_name
- abc assert
- abc describe

main Tests ✓ 0 × 0 coverage: progress You, now Ln 13, Col 102 Spaces: 2 UTF-8 LF {} JavaScript

Debugging in VS Code

The screenshot shows the VS Code interface during a debugging session. The top toolbar includes a 'RUN AND DEBUG' button and a 'No Configurations' dropdown. The left sidebar contains icons for Explorer, Search, Source Control, Run and Debug, and Extensions. The main editor area displays the code for the `isEmail` function in `isEmail.js`. The function is currently paused at a breakpoint on line 99, which is highlighted in yellow. The current value of `display_name` is `'Philipp Salvisberg'`. The bottom panel shows the 'TERMINAL' tab with the following output:

```
Debugger attached.

increase_salary
  ✓ By 0 percent for dept 10 (400ms)

util
  toEpoch()
    ✓ Convert today to Unix Time

Validator
  isEmail()
    ✓ philipp@salvis.com is valid
    ✓ Philipp Salvisberg <philipp@salvis.com> is invalid with default options
```

The status bar at the bottom indicates the current position is Ln 99, Col 38, with 2 spaces, UTF-8 encoding, and LF line endings. The file is identified as JavaScript.

MLE Post-Execution Debugging

```
1 set serveroutput on size unlimited
2 declare
3   co_breakpoints constant json := json(q'~
4 {
5   "version": "1.0",
6   "debugpoints": [
7     {
8       "at": {"name": "CREATE_TEMP_TABLE_MOD", "line": 4},
9       "actions": [{"type": "watch", "id": "tableName"}],
10      "condition": "tableName.includes('-')"
11    },
12    {
13      "at": {"name": "CREATE_TEMP_TABLE_MOD", "line": 12},
14      "actions": [{"type": "snapshot"}]
15    }
16  ]
17 }
18 ~');
19   l_sink          blob;
20   l_output        json;
21 begin
22   dbms_lob.createtemporary(l_sink, false, dbms_lob.call);
23   dbms_mle.enable_debugging(co_breakpoints, l_sink);
24   ut.run('test_create_temp_table.create_invalid_temp_table_js');
25   l_output := dbms_mle.parse_debug_output(l_sink);
26   dbms_output.put_line('MLE debug output: '
27     || chr(10)
28     || json_serialize(l_output returning clob pretty));
29   dbms_mle.disable_debugging();
30 end;
31 /
```

```
1 all
2   test_create_temp_table
3     js
4       create_invalid_temp_table_js [.039 sec]
5
6 Finished in .043893 seconds
7 1 tests, 0 failed, 0 errored, 0 disabled, 0 warning(s)
8
9 MLE debug output:
10 [
11   [
12     {
13       "at" :
14         {
15           "name" : "DEMO.CREATE_TEMP_TABLE_MOD",
16           "line" : 4
17         },
18       "values" :
19         {
20           "tableName" : "TEST-JS"
21         }
22     }
23   ]
24 ]
25
26
27 PL/SQL procedure successfully completed.
```

Debugging in Database Actions

The screenshot displays the Oracle Database Actions IDE interface. At the top, the breadcrumb navigation shows 'ORACLE Database Actions | MLE JS'. A search bar is located in the top right corner. The main workspace is divided into three panels: a Navigator on the left, an Editor in the center, and a Debug Console on the right.

Navigator: Shows a tree view of the project structure. The 'DEMO' environment is selected. Under 'Modules', several modules are listed, including 'CREATE_TEMP_TABLE_MOD', 'INCREASE_SALARY_LOOP_MOD', 'INCREASE_SALARY_MOD', 'JIMP_MOD', 'SENTIMENT_MOD', 'SQL_TEMPLATE_TAG_MOD', 'UTIL_MOD', 'VALIDATOR_MOD', and 'VALIDATOR_API'. The 'CREATE_TEMP_TABLE_MOD' module is expanded, showing its 'Environments' (DEMO_ENV) and 'Call Specifications' (CREATE_TEMP_TABLE_JS).

Editor: Displays a JavaScript snippet named 'snippet_debug_create_temp_table*'. The code is as follows:

```
1 (async() => {
2   const ctt = await import('create_temp_table');
3   ctt.createTempTable('ok');
4 })();
```

The 'Environment' is set to 'DEMO_ENV' and the 'Debug Specification' is 'create_temp_table_mod_debug'. The code is currently running, as indicated by the green play button icon.

Debug Console: Shows the output of the debug session. The selected snippet is '"create_temp_table" - line 12'. The output is:

```
result: {"metaData":{"name":"TAB"},"rows":{"TAB":"ok"}}
conn: {"_parameters":{"_extendedMetaData":false,"_fetchArraySize":10,"_fetchAsPsqlWrapper":[],"_fetch/
this: {}
tableName: ok
```

Code Editor: Shows the source code for the 'createTempTable' function. The code is as follows:

```
1 import oracledb from "mle-js-oracledb";
2
3 export function createTempTable(tableName) {
4   const conn = oracledb.defaultConnection();
5
6   // may throw a "ORA-04161: Database Error" without reference to this module (bad)
7   const result = conn.execute(
8     `select dbms_assert.simple_sql_name(:tableName) as tab`,
9     [tableName]
10  );
11
12  conn.execute(
13    `create private temporary table ora${pvt}_${result.rows[0].TAB} (id number)`
14  );
15 }
16
17 export function createTempTable2(tableName) {
18   const conn = oracledb.defaultConnection();
19
20   // may throw a "ORA-44003: invalid SQL name" with reference to this module (good)
21   const result = conn.execute(
22     `begin
23       :tab := dbms_assert.simple_sql_name(:tableName);
24     end;`,
25     {tab: {dir: oracledb.BIND_OUT}, tableName}
26  );
```

The code is color-coded, and a blue dot on line 12 indicates the current execution point.

Footer: The bottom status bar shows '2' errors, '0' warnings, and '0' info messages. The message reads: '1:37:09 PM - REST call resolved successfully.' The interface is powered by ORDS.

Key Messages

Pros & Cons

PL/SQL



Pros

- Faster runtime
- Lower SQL injection risk
- Lower memory consumption
- All data types are supported
- Fastest startup times



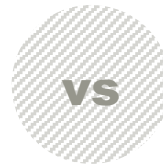
Open

- Compile-time dependencies (static SQL)



Cons

- Less ready-to-use 3rd party libs
- Rudimentary ecosystem
- Less popular, more difficult to find devs
- Slower evolution – feels old



JavaScript



Pros

- More ready-to-use 3rd party libs [npm](#)
- Excellent ecosystem
- More popular, easier to find devs
- Faster evolution – feels modern
- Fast startup times



Open

- Runtime dependencies (dynamic SQL)



Cons

- Slower runtime (gap is disappointing)
- Higher SQL injection risk
- Higher memory consumption
- No support for long, long raw, xmltype, object types, bfile, ref cursor

Is Tom Kyte's Mantra Still Valid?

"I have a pretty simple mantra when it comes to developing database software, one that has been consistent for many years:

- You should do it in a single **SQL** statement if at all possible. And believe it or not, it is almost always possible. This statement is even truer as time goes on. SQL is an extremely powerful language.
- If you can't do it in a single SQL Statement, do it in **PL/SQL**—as little PL/SQL as possible! Follow the saying that goes "more code = more bugs, less code = less bugs."
- If you can't do it in PL/SQL, try a **Java** stored procedure. The times this is necessary are extremely rare nowadays with Oracle9i and above. PL/SQL is an extremely competent, fully featured 3GL.
- If you can't do it in Java, do it in a **C** external procedure. This is most frequently the approach when raw speed or using a third-party API written in C is needed.
- If you can't do it in a C external routine, you might want to seriously think about why it is you need to do it. "

-- Tom Kyte, *Expert Oracle Database Architecture, Third Edition, 2014, page 13*

1 Consider the [technical dept.](#)

2 If you can't do it in a single SQL Statement, do it in PL/SQL or JavaScript...



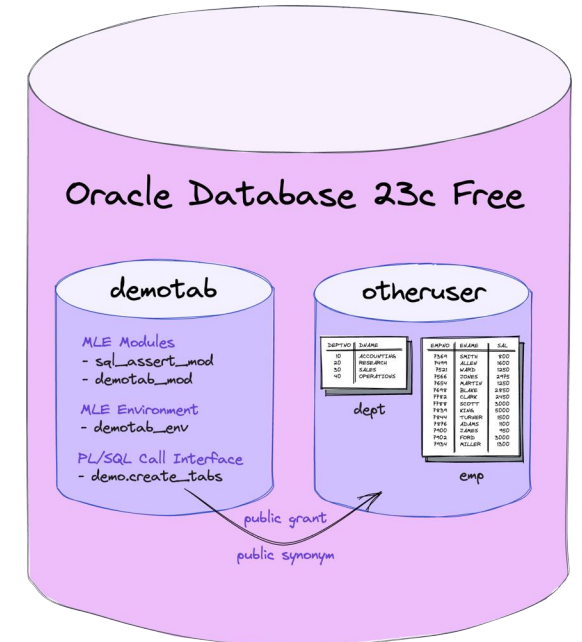
Welcome JavaScript in the Oracle Database

- **Excellent for ...**

- Reusing existing algorithms to process data
- Example npm modules:
 - [validator](#) (isMail, isEAN, isIBAN, isCreditCard, isHash, ...)
 - [sentiment](#) (sentiment analysis of arbitrary text)
 - [jimp](#) (image processing as replacement for Oracle Multimedia)
 - [sql-assert](#) (alternative for dbms_assert to avoid SQL injection)

- **However, ...**

- Develop and test MLE modules outside of the DB
- Avoid the use of DBMS_MLE whenever possible
- Use connection pools and monitor resource usage
- JavaScript is not for everything ...
... consider using DB features before reinventing things



Source: <https://www.salvis.com/blog/2023/11/12/mle-typescript-javascript-modules/>



Thank You