

# PL/SQL vs. JavaScript in Oracle Database 23ai

Philipp Salvisberg  
18<sup>th</sup> October 2024



# Welcome



## Philipp Salvisberg

Founder, Owner and CEO of Grisselbav GmbH

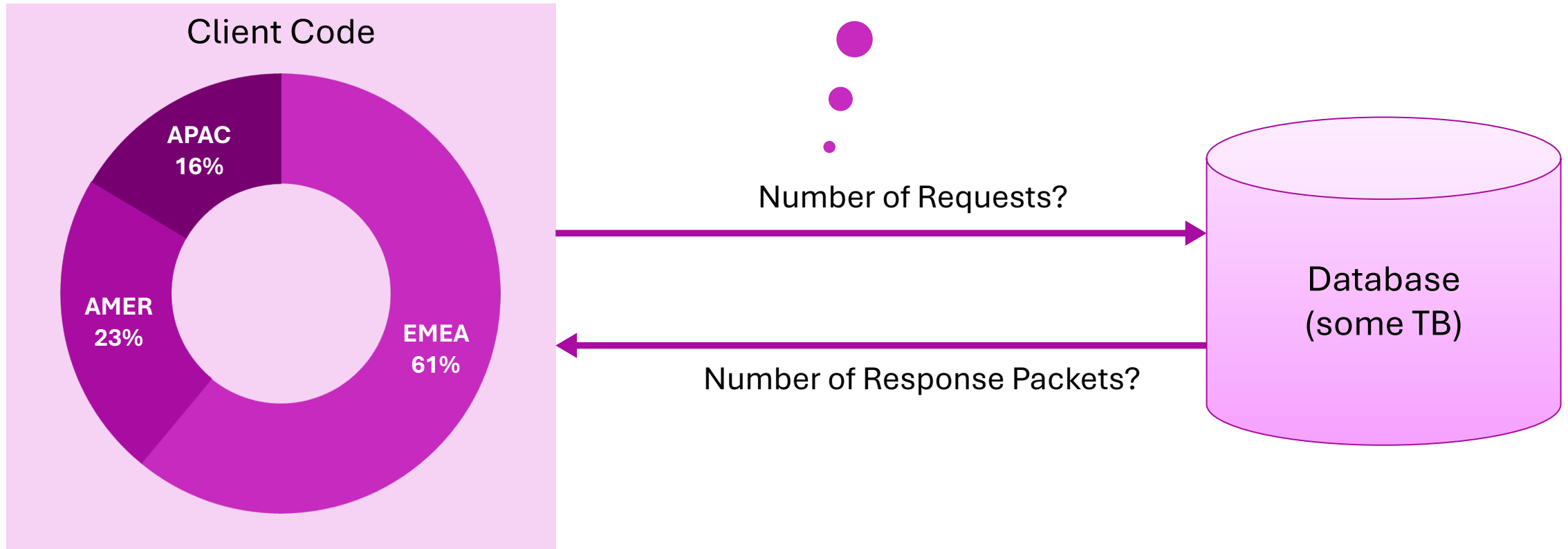
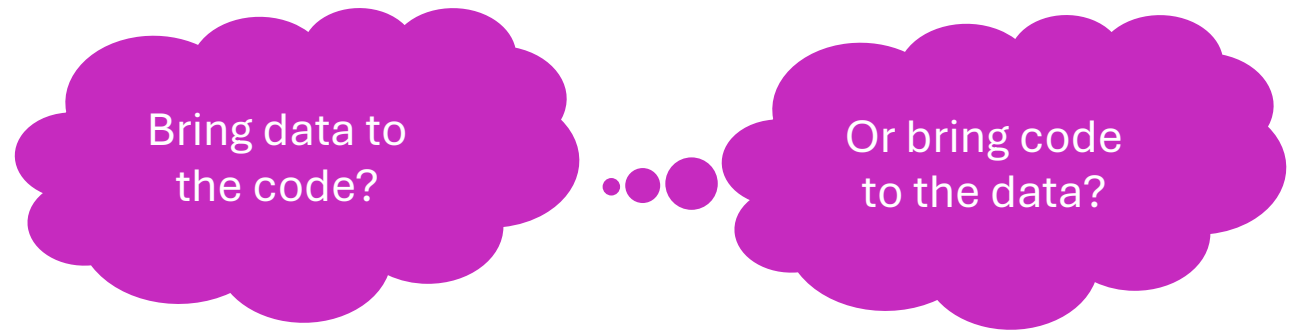
- Database-centric Development
- Model-driven Software Development
- Open-Source Development

[philipp.salvisberg@grisselbav.com](mailto:philipp.salvisberg@grisselbav.com)

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

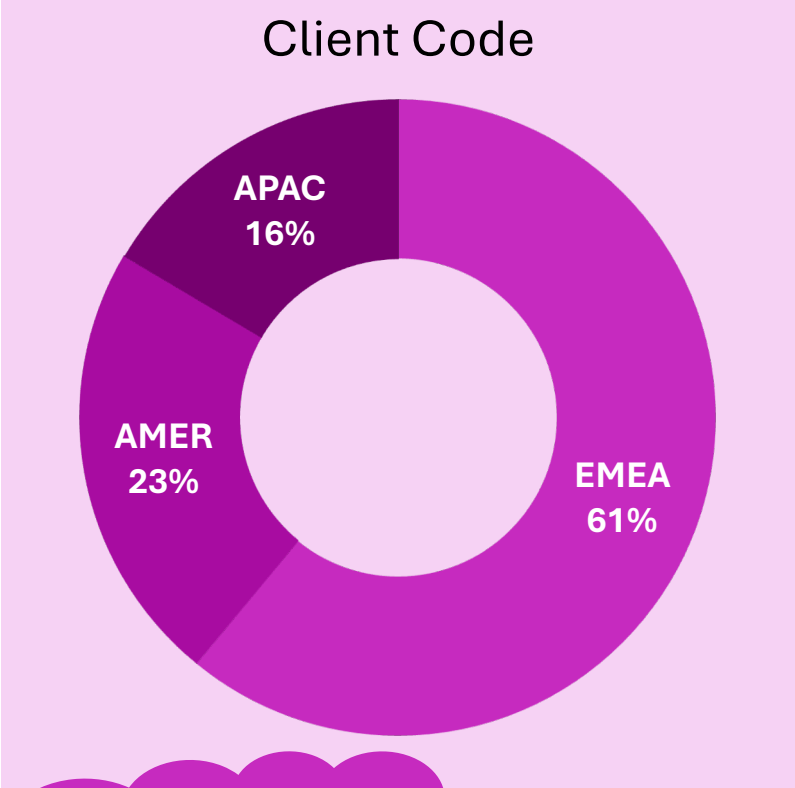
# Why Do We Need Code in the Database?

# Processing Data



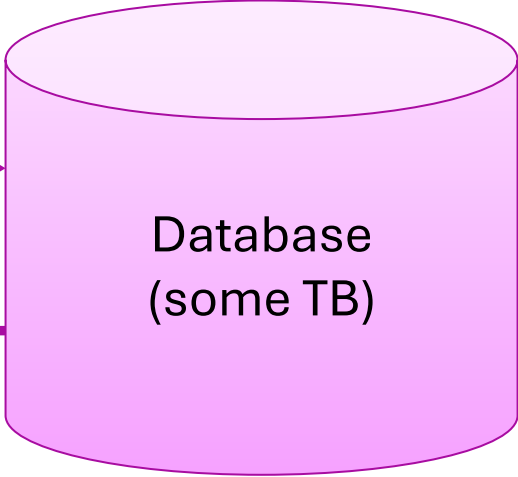
# Bring Code to the Data

Minimize communication for good performance



One Request

One Response Packet



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

```
create or replace procedure utl_xml_parse_query(
  in_current_userid in          number,
  in_schema_name   in          varchar2,
  in_query         in          clob,
  io_result        in out nocopy clob
) is
  language c
  library sys.utl_xml_lib          -- static lib, part of the oracle binary
  name "kuxParseQuery"           -- name of the C function in the lib
  with context parameters (
    context,                      -- connection context, allows callbacks
    in_current_userid ocinumber,   -- content
    in_current_userid indicator,   -- null indicator
    in_schema_name   ocistring,    -- content
    in_schema_name   indicator,    -- null indicator
    in_query         ociloblocator, -- content
    in_query         indicator,    -- null indicator
    io_result        ociloblocator, -- content
    io_result        indicator     -- null indicator
  );
/
```

# PL/SQL (Version 7.0)

```
create or replace procedure increase_salary_plsql(  
    in_deptno    in number,  
    in_by_percent in number  
) is  
begin  
    update emp  
        set sal = sal + sal * in_by_percent / 100  
        where deptno = in_deptno;  
end increase_salary_plsql;  
/
```

Usages in the  
data dictionary

Compile errors &  
no SQLi

Static  
SQL



# PL/SQL – Without Data Processing (DML)?

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

Is it necessary that we implement that ourselves?

# Java (Version 8.1)

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

PL/SQL call spec for  
accessibility and  
data type conversions

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

# Java – Using 3rd Party Libraries

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

Must be compatible with  
the OJVM in the database

1.8.0\_371 in 19.19  
11.0.24 in 23.5

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

# Java – Permissions

```
declare
  co_user constant all_users.username%type := 'MYUSER';
begin
  dbms_java.grant_permission(co_user,
    'SYS:java.net.SocketPermission', '*:1024-65535', 'connect, resolve');
  dbms_java.grant_permission(co_user,
    'SYS:java.io.FilePermission', '/tmp', 'read, write');
  dbms_java.grant_permission(co_user,
    'SYS:java.lang.RuntimePermission', 'getClassLoader', '');
  dbms_java.grant_permission(co_user,
    'SYS:java.lang.RuntimePermission', 'setContextClassLoader', '');
end;
/
```

# Dynamic JavaScript (Version 21)

```
create or replace function to_epoch_djs(in_ts in timestamp) return number is
  co_js constant clob := q'~
    const bindings = require("mle-js-bindings");
    const ts = bindings.importValue("ts");
    bindings.exportValue("millis", ts.valueOf());
  ~';
  l_ctx      dbms_mle.context_handle_t;
  l_millis number;
begin
  l_ctx := dbms_mle.create_context();
  dbms_mle.export_to_mle(l_ctx, 'ts', in_ts);
  dbms_mle.eval(l_ctx, 'JAVASCRIPT', co_js);
  dbms_mle.import_from_mle(l_ctx, 'millis', l_millis);
  dbms_mle.drop_context(l_ctx);
  return l_millis;
end to_epoch_djs;
/
```

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

Map input and output  
types dynamically

# Static JavaScript (Version 23)

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

PL/SQL call spec for  
accessibility and  
optional data type conversions

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

# Inline MLE Call Specification

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

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

```
create or replace function to_epoch_js2("in_ts" in timestamp)
return number is
mle language javascript ' return in_ts.valueOf();';
/
```

JavaScript function

# SQL in JavaScript

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

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

```
SQLcl
-- registers mle as SQLcl command (optional)
script https://raw.githubusercontent.com/PhilippSalvisberg/mle-sqlcl/main/mle.js register
-- create or replace mle module validator_mod language javascript version 13.12.0 as ...
mle install validator_mod https://esm.run/validator@13.12.0 13.12.0

-- validator_mod is a single object in the Oracle Data Dictionary
select module_name, version, language_name, length(module) from user_mle_modules;
```

MODULE_NAME	VERSION	LANGUAGE_NAME	LENGTH(MODULE)
VALIDATOR_MOD	13.12.0	JAVASCRIPT	123260

Size in bytes of  
minimized library  
from NPM

# JavaScript – MLE Module Wrapper

```
spec.sql
create or replace package validator_api is
  function is_email(in_email in varchar2)
    return boolean deterministic;
end validator_api;
/
```

Hidden MLE  
Call Spec in the  
package body!

```
body.sql
create or replace package body validator_api is
  function is_email_internal(
    in_email in varchar2,
    in_options in json
  ) return boolean deterministic as mle module
  • validator_mod signature 'default.isEmail(string, any)';

  function is_email(in_email in varchar2)
    return boolean deterministic is
  begin
    return is_email_internal(
      in_email => in_email,
      in_options => json(
        {"allow_display_name": false,
         "allow_undescodes": 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": ["dubious.com"],
         "host_whitelist": []}));
  end is_email;
end validator_api;
```

# JavaScript – MLE Environments

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

Modules that  
can be imported  
from JavaScript

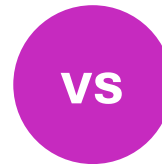
Can handle  
different module  
versions

Compile options

# Languages in the Oracle Database 23ai

## 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)

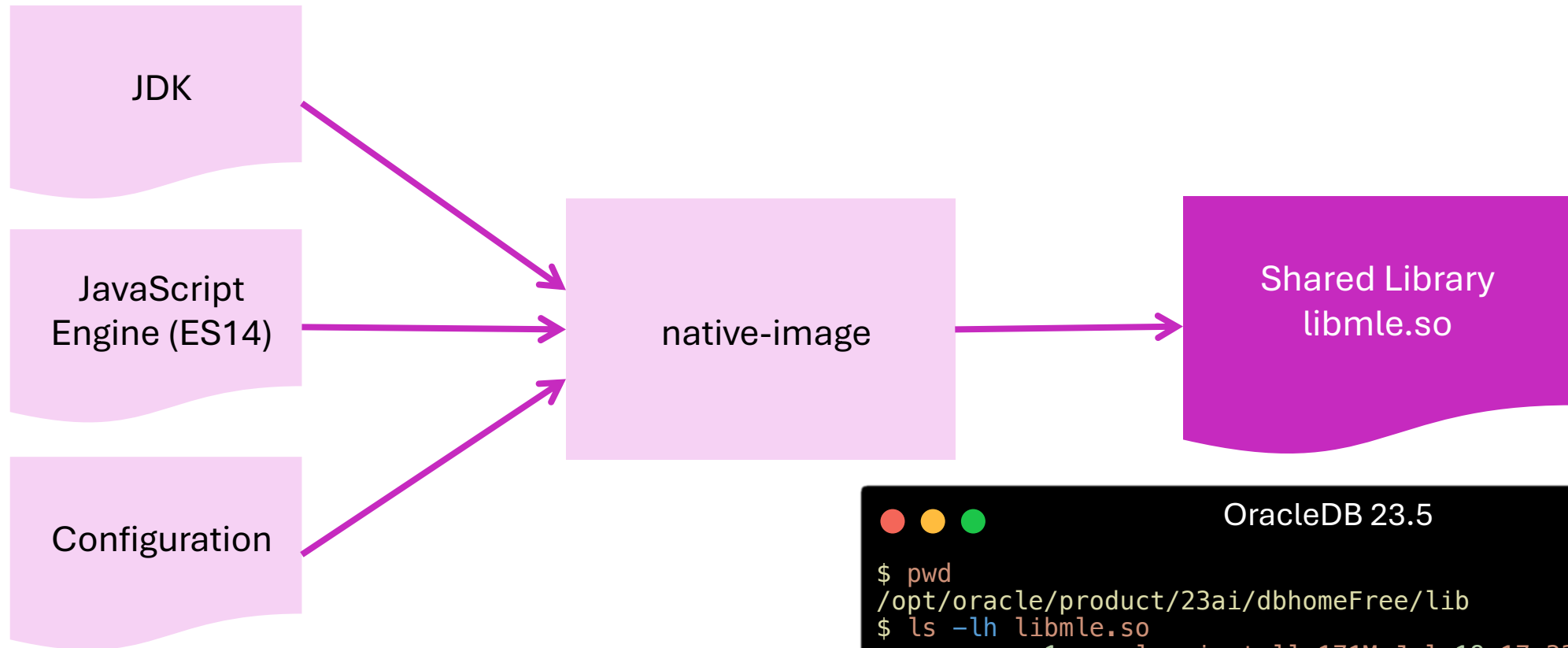
## OracleDB 19.19

```
$ 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
```

## OracleDB 23.5

```
$ pwd
/opt/oracle/product/23ai/dbhomeFree/javavm
$ du -h | sort -k2
228M     .
20K      ./admin
116K     ./conf
24K      ./conf/management
84K      ./conf/security
44K      ./doc
748K     ./install
28K      ./install/sbs
222M     ./jdk
222M     ./jdk/jdk11
185M     ./jdk/jdk11/admin
38M      ./jdk/jdk11/lib
128K     ./jdk/jdk11/lib/security
1.7M     ./lib
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



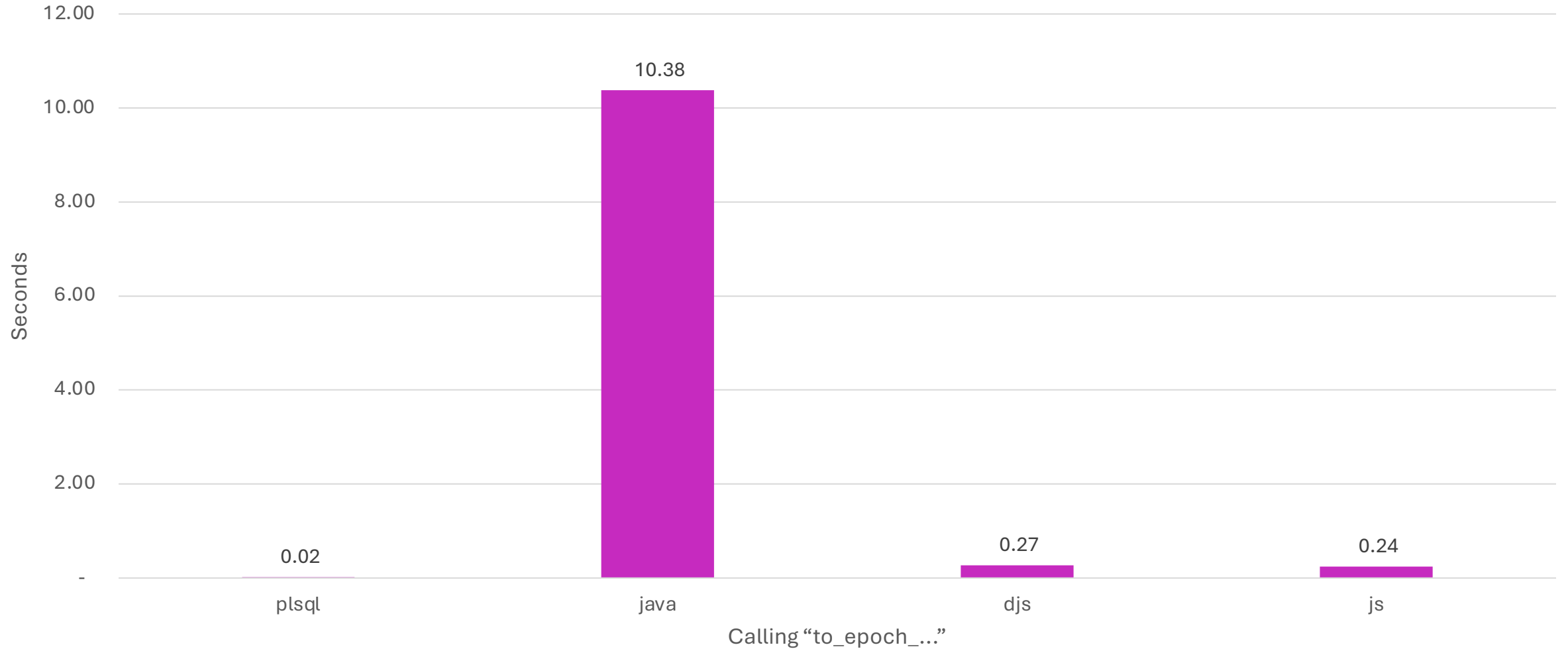
```
OracleDB 23.5
$ pwd
/opt/oracle/product/23ai/dbhomeFree/lib
$ ls -lh libmle.so
-rw-r----- 1 oracle oinstall 171M Jul 18 17:33 libmle.so
```

# Comparing Performance & Resource Usage

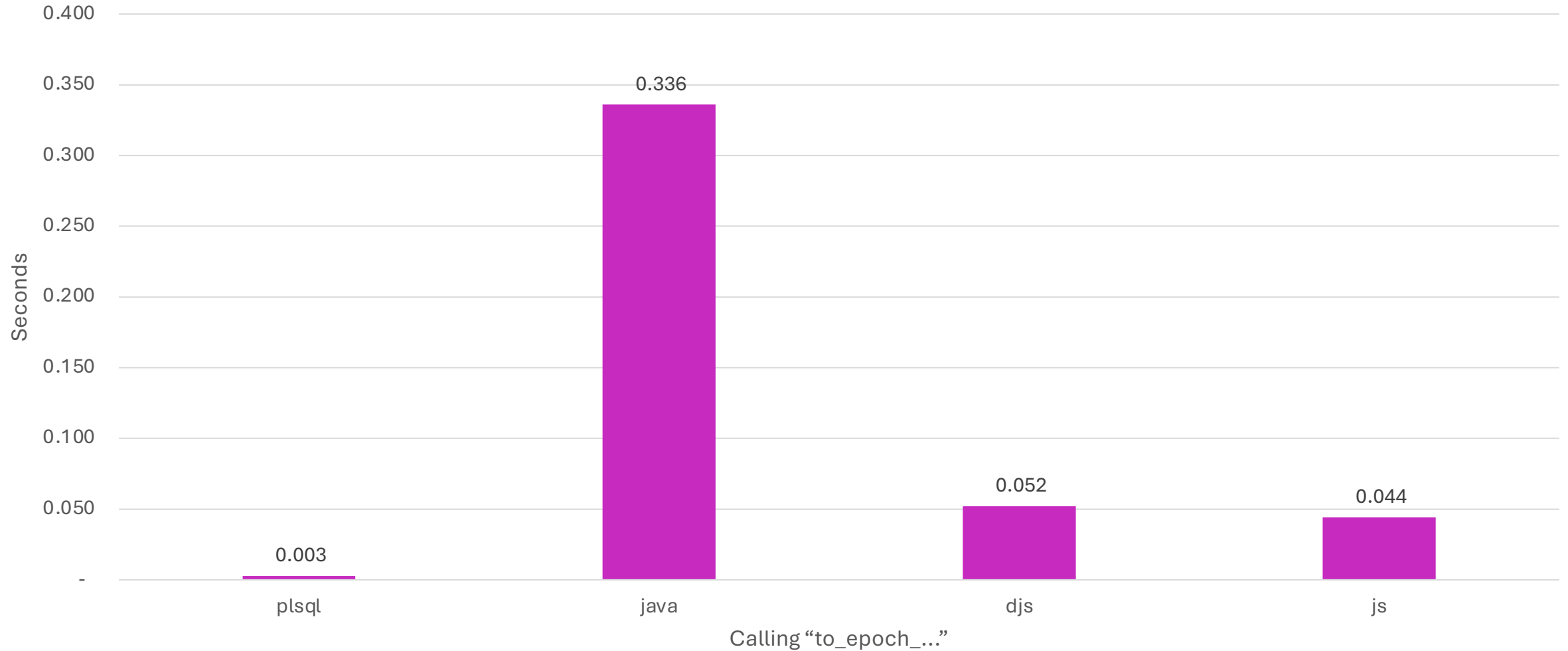
## Best of 5 Attempts



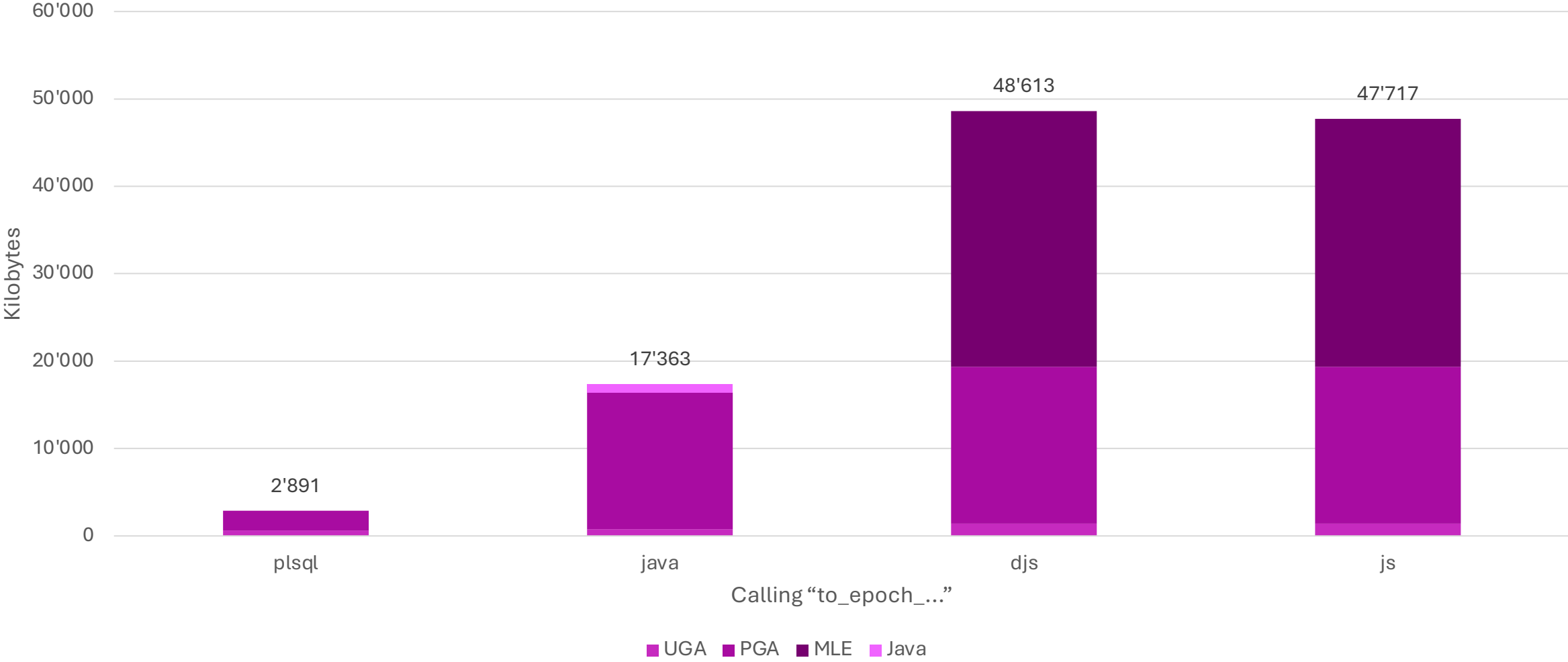
# Runtime of First Call after DB Restart



# Runtime of First Call in New DB Session



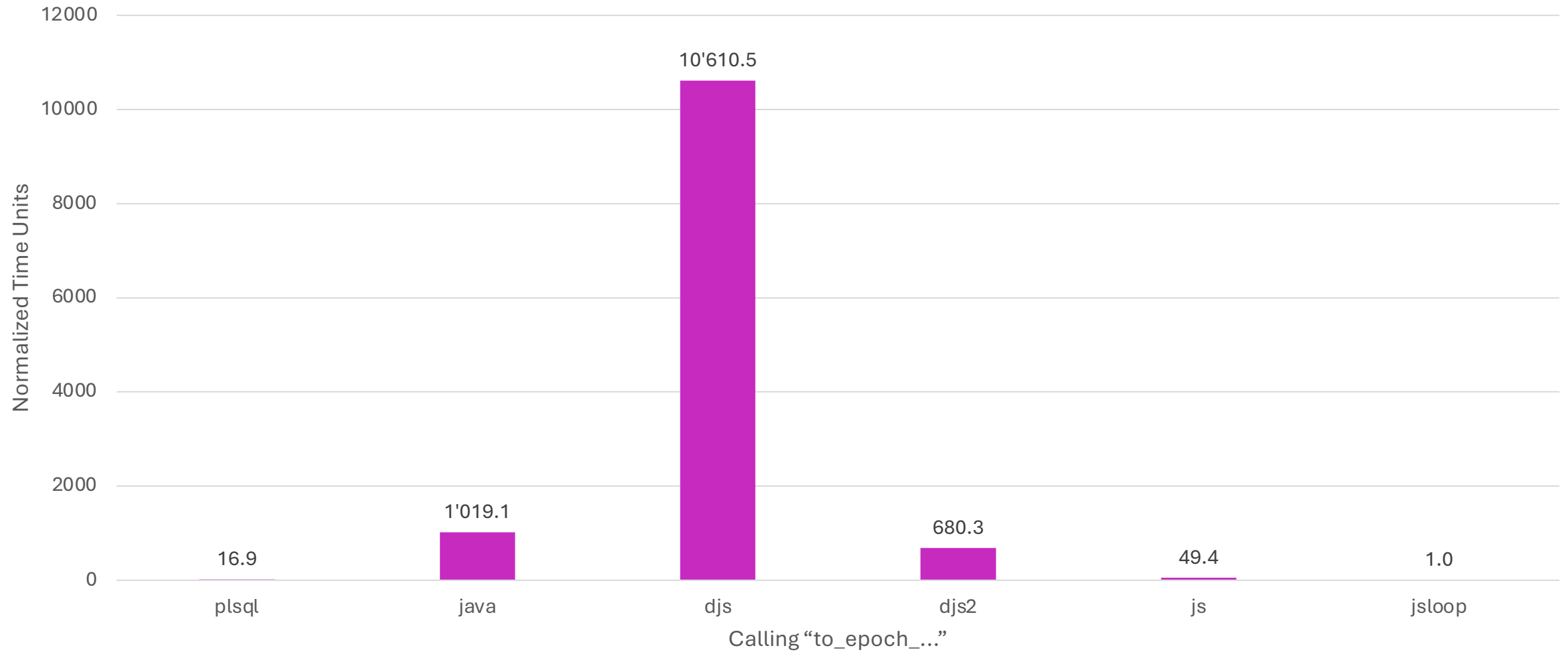
# Max. Memory Usage After Single Call



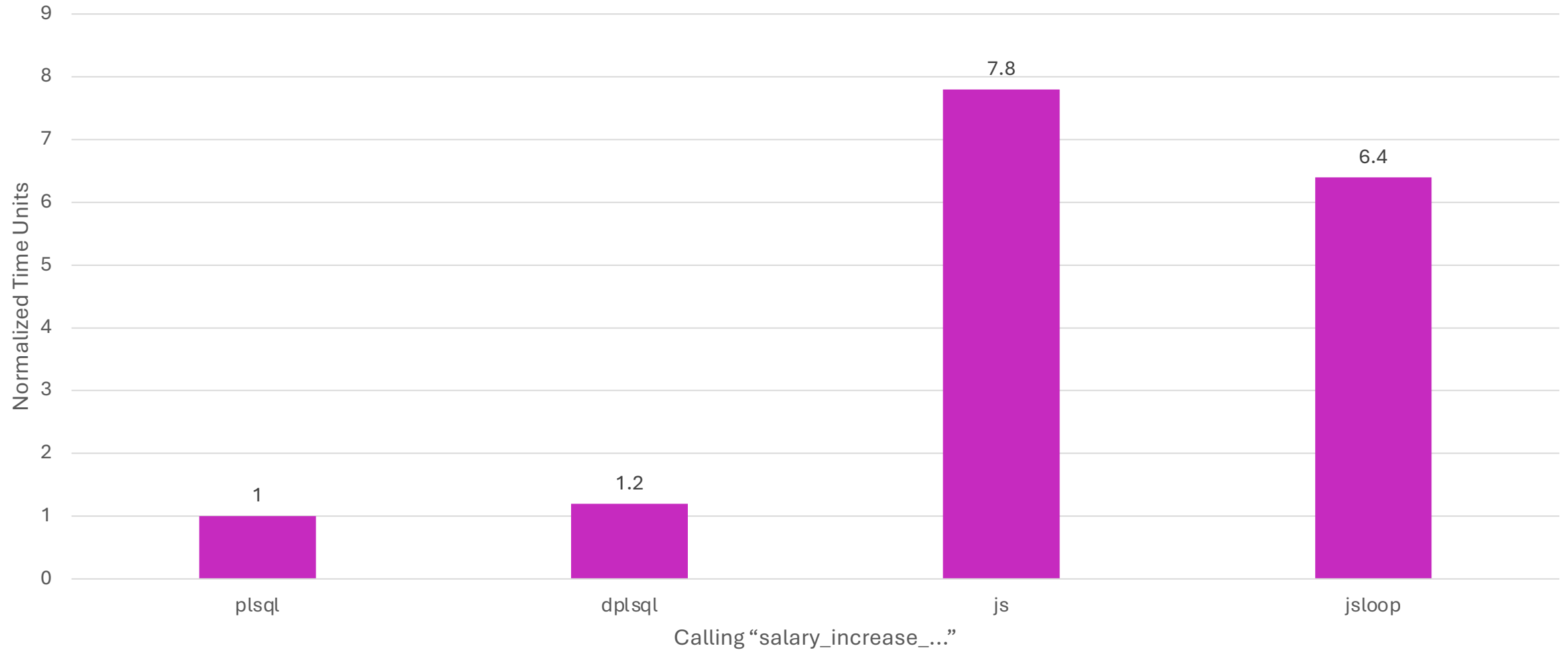
# Comparing Apples with Pears?



# Runtime of 100K Calls of to\_epoch



# Runtime of 100K Calls of salary\_increase



# Good Security Practices & Default Exceptions

# Binds and Assertions

## JavaScript

```
create or replace mle module
  create_temp_table_mod language javascript as

export function createTempTable(tableName) {
  const result = session.execute(
    `select dbms_assert.simple_sql_name(
      :tableName
    ) as tab`,
    [tableName]
  );

  session.execute(
    `create private temporary table
      ora\${ptt_${result.rows[0].TAB}} (id number)`
  );
}
```

## PL/SQL

```
create or replace procedure create_temp_table_plsql
  (in_table_name in varchar2)
is
  co_tmpl constant varchar2(1000 char) :=
    'create private temporary table
      ora${ptt_#valid_table_name#} (id number)';
begin
  execute immediate replace(co_tmpl,
    '#valid_table_name#',
    dbms_assert.simple_sql_name(in_table_name));
end;
```

Template literals and JSON  
integration are great



# Assert in JavaScript

## DDL

```
create or replace mle module
  create_temp_table_mod language javascript as
import { simpleSqlName } from "sql-assert";
export function createTempTable(tableName) {
  session.execute(
    `create private temporary table
      ora\${ptt_}\${simpleSqlName(tableName)} (id number)`
  );
}
/

create or replace procedure create_temp_table_js(
  in_table_name in varchar2
) as
mle module create_temp_table_mod
env demo_env
signature 'createTempTable(string)';
/
```

## Run in SQLcl

```
SQL> exec create_temp_table_js('my-table');

Error starting at line : 1 in command -
BEGIN create_temp_table_js('my-table'); END;
Error report -
ORA-04161: Error: Invalid SQL name.
ORA-04171: at e (DEM01.SQL_ASSERT_MOD:7:397)
ORA-06512: at "DEM01.CREATE_TEMP_TABLE_JS", line 1
ORA-06512: at line 1
04161. 00000 - "%s"
*Cause:      A runtime error occurred while evaluating a Multilingual Engine
              (MLE) code snippet or call specification.
*Action:     Fix the MLE language code that causes the runtime error. Use the
              source location reported in the error message to identify the code
              that needs to be fixed. Use the DBMS_MLE.get_error_stack or
              DBMS_MLE.get_ctx_error_stack functions to retrieve the MLE language
              stack trace for the error.

More Details :
https://docs.oracle.com/error-help/db/ora-04161/
https://docs.oracle.com/error-help/db/ora-04171/
https://docs.oracle.com/error-help/db/ora-06512/
```

Good ORA-04161 message,  
get missing references via  
dbms\_mle.get\_error\_stack...

# Get Missing Error Stack

## Run in SQLcl

```
SQL> exec create_temp_table_js('my-table');

Error starting at line : 1 in command -
BEGIN create_temp_table_js('my-table'); END;
Error report -
ORA-04161: Error: Invalid SQL name.
ORA-04171: at e (DEM01.SQL_ASSERT_MOD:7:397)
ORA-06512: at "DEM01.CREATE_TEMP_TABLE_JS", line 1
ORA-06512: at line 1
04161. 00000 - "%s"
*Cause:      A runtime error occurred while evaluating a Multilingual Engine
              (MLE) code snippet or call specification.
*Action:     Fix the MLE language code that causes the runtime error. Use the
              source location reported in the error message to identify the code
              that needs to be fixed. Use the DBMS_MLE.get_error_stack or
              DBMS_MLE.get_ctx_error_stack functions to retrieve the MLE language
              stack trace for the error.

More Details :
https://docs.oracle.com/error-help/db/ora-04161/
https://docs.oracle.com/error-help/db/ora-04171/
https://docs.oracle.com/error-help/db/ora-06512/
```



John McEnroe, June 22, 1981

"You cannot  
be serious!"

## Run in SQLcl after error

```
set serveroutput on size unlimited
declare
  t_frames dbms_mle.error_frames_t;
begin
  t_frames := dbms_mle.get_error_stack(
              module_name => 'CREATE_TEMP_TABLE_MOD',
              env_name     => 'DEMO_ENV'
              );
  for i in t_frames.count
  loop
    dbms_output.put_line(
      'ORA-04171: at '
      || t_frames(i).func
      || '('
      || t_frames(i).source
      || ':'
      || t_frames(i).line
      || ':'
      || t_frames(i).col
      || ')'
    );
  end loop;
end;
/

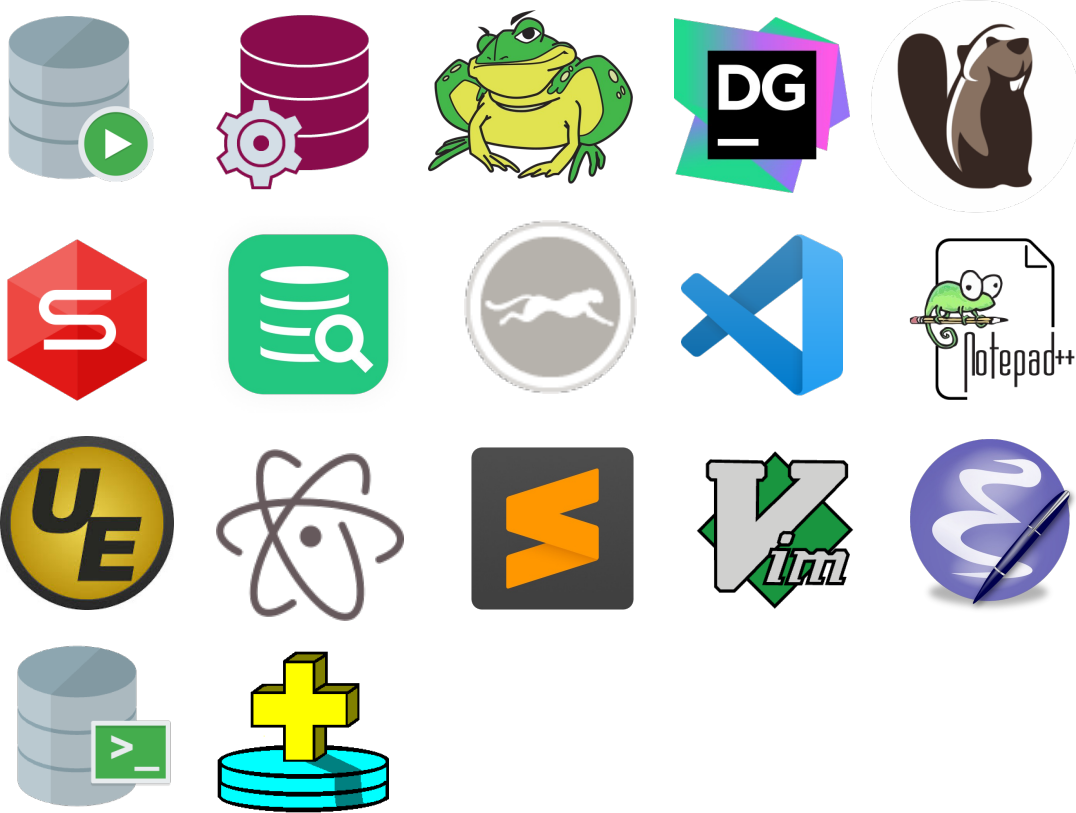
ORA-04171: at createTempTable (DEM01.CREATE_TEMP_TABLE_MOD:6:20)

PL/SQL procedure successfully completed.
```

# 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?
- allow\_display\_name
- assert
- describe

main coverage: progress You, now Ln 13, Col 102 Spaces: 2 UTF-8 LF JavaScript

# Debugging in VS Code

The screenshot displays the Visual Studio Code interface during a debugging session. The top bar shows the file path: `js23c`. The left sidebar contains several panels:

- VARIABLES:** Shows the current state of variables. `options` is expanded to show properties like `allow_display_name: true`, `allow_underscores: false`, and `require_display_name: false`.
- WATCH:** Currently empty.
- CALL STACK:** Shows the execution path: `Node.js Process: npm [59456]` (RUNNING) and `mocha [59500]` (PAUSED ON BREAKPOINT). The current frame is `Object.isEmail` at `sandbox/node_modules/valid...`.
- LOADED SCRIPTS:** Shows the loaded script `demo-validator.js` at `sandbox/test`.
- BREAKPOINTS:** Shows two breakpoints: `Caught Exceptions` and `Uncaught Exceptions`.

The main editor displays the `isEmail.js` file. The code is as follows:

```
79 function isEmail(str, options) {
80   (0, _assertString.default)(str);
81   options = (0, _merge.default)(options, default_email_options);
82
83   if (options.require_display_name || options.allow_display_name) {
84     var display_email = str.match(splitNameAddress);
85
86     if (display_email) {
87       var display_name = display_email[1]; // Remove display name and angle brackets to get email address
88       // Can be done in the regex but will introduce a ReDOS (See #1597 for more info)
89
90       str = str.replace(display_name, '').replace(/(<|>)/g, ''); // sometimes need to trim the last space to get the display name
91       // because there may be a space between display name and email address
92       // eg. myname <address@gmail.com>
93       // the display name is `myname` instead of `myname `, so need to trim the last space
94
95       if (display_name.endsWith(' ')) {
96         display_name = display_name.slice(0, -1);
97       }
98       // Philipp Salvisberg
99       if (!validateDisplayName(display_name)) {
100         return false;
101       }
102     }
103     // also if (options.require_display_name) {
```

The breakpoint is set at line 99. The terminal shows 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: `Ln 99, Col 38`, `Spaces: 2`, `UTF-8`, `LF`, and `JavaScript`.

# MLE Post-Execution Debugging

## Run test with debug output

```
set serveroutput on size unlimited
declare
  co_breakpoints constant json := json(q'~
{
  "version": "1.0",
  "debugpoints": [
    {
      "at": {"name": "CREATE_TEMP_TABLE_MOD", "line": 5},
      "actions": [{"type": "watch", "id": "tableName"}],
      "condition": "tableName.includes('-')"
    },
    {
      "at": {"name": "CREATE_TEMP_TABLE_MOD", "line": 10},
      "actions": [{"type": "snapshot"}]
    }
  ]
}
~');
  l_sink          blob;
  l_output        json;
begin
  sys.dbms_lob.createtemporary(
    lob_loc => l_sink, cache => false, dur => sys.dbms_lob.call);
  sys.dbms_mle.enable_debugging(debugspec => co_breakpoints, sink => l_sink);
  ut.run('DEM01:all.test_create_temp_table.nested_context_#2' -- js context
    || '.create_invalid_temp_table_js');
  l_output := sys.dbms_mle.parse_debug_output(l_sink);
  sys.dbms_output.put_line('MLE debug output: '
    || chr(10)
    || json_serialize(l_output returning clob pretty));
  sys.dbms_mle.disable_debugging();
end;
/
```

## Output

```
all
  test_create_temp_table
    js
      create_invalid_temp_table_js [.028 sec]

Finished in .030085 seconds
1 tests, 0 failed, 0 errored, 0 disabled, 0 warning(s)

MLE debug output:
[
  [
    {
      "at" :
      {
        "name" : "DEM01.CREATE_TEMP_TABLE_MOD",
        "line" : 5
      },
      "values" :
      {
        "tableName" : "TEST-JS"
      }
    }
  ]
]

PL/SQL procedure successfully completed.
```

# Debugging in Database Actions

The screenshot displays the Oracle Database Actions interface. The top navigation bar shows "ORACLE Database Actions | JavaScript" and a search bar. The main interface is divided into several sections:

- Navigator:** Shows the current environment "DEMO1" and a list of modules including "CREATE\_TEMP\_TABLE\_MOD".
- Snippets:** A list of snippets is shown, with "CREATE\_TEMP\_TABLE\_JS" selected.
- Editor:** Displays the JavaScript code for the snippet. The code is as follows:

```
1 // snippet
2 (async() => {
3   const ctt = await import('create_temp_table');
4   ctt.createTempTable('ok');
5 }());
```
- Debug Console:** Shows the execution result for the snippet. The output is:

```
result:{"metaData":{"name":"TAB"},"rows":{"TAB":"ok"}}
this:{}
tableName:ok
```
- Code Editor (Bottom):** Shows the source code for the "createTempTable" function, which is used by the snippet. The code is as follows:

```
1 import { simpleSqlName } from "sql-assert";
2
3 export function createTempTable(tableName) {
4   // may throw a "ORA-04161: Database Error" without reference to this module (bad)
5   const result = session.execute(
6     `select dbms_assert.simple_sql_name(:tableName) as tab`,
7     [tableName]
8   );
9
10  session.execute(
11    `create private temporary table ora\${ptt}\${result.rows[0].TAB} (id number)`
12  );
13 }
14
15 export function createTempTable2(tableName) {
16   // may throw a "ORA-44003: invalid SQL name" with reference to this module (good)
17   const result = session.execute(
18     `begin
19       :tab := dbms_assert.simple_sql_name(:tableName);
20     end;`,
21     {tab: {dir: oracledb.BIND_OUT}, tableName}
22   );
23
24   session.execute(
25     `create private temporary table ora\${ptt}\${result.outBinds.tab} (id number)`
26   );
27 }
```

The status bar at the bottom indicates "12:36:53 PM - REST call resolved successfully." and "Powered by ORDS".



# Key Messages

# Pros & Cons

## PL/SQL

### 😊 Pros

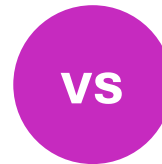
- Faster runtime (with embedded SQL)
- Lower SQL injection risk
- Lower memory consumption
- All data types are supported
- Fastest startup times

### 🤔 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 (with embedded SQL)
- 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 3*

1 Consider the [technical dept.](#)

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

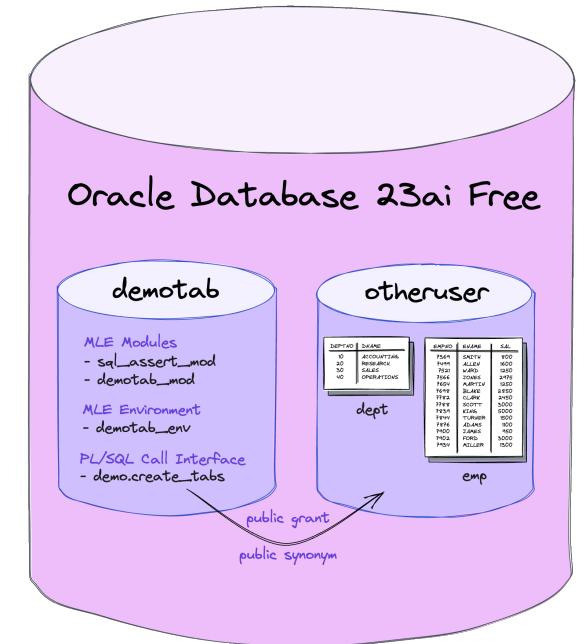
# Welcome JavaScript in Oracle Database

- **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!

