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

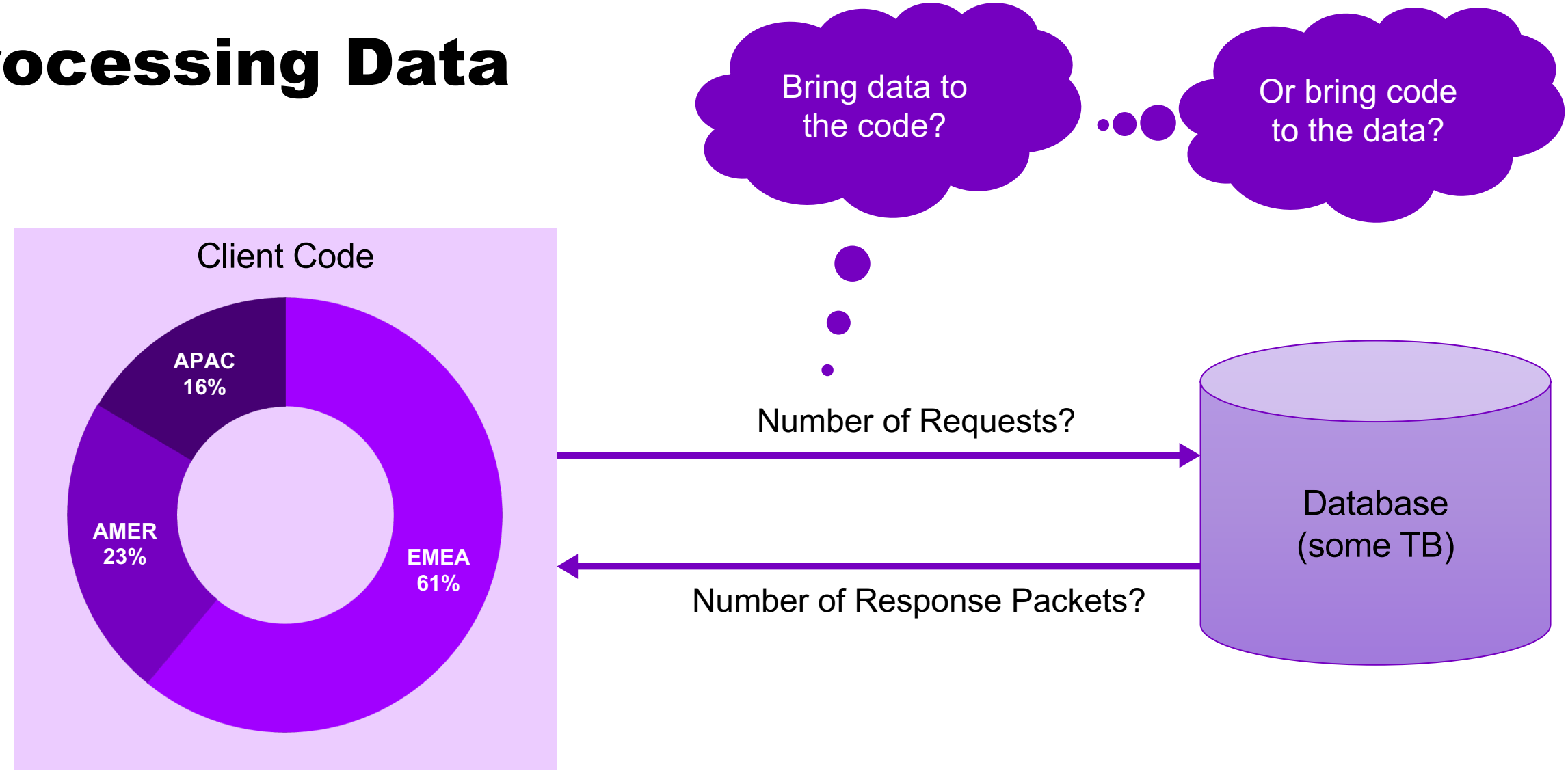
philipp.salvisberg@accenture.com

https://www.salvis.com/blog

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

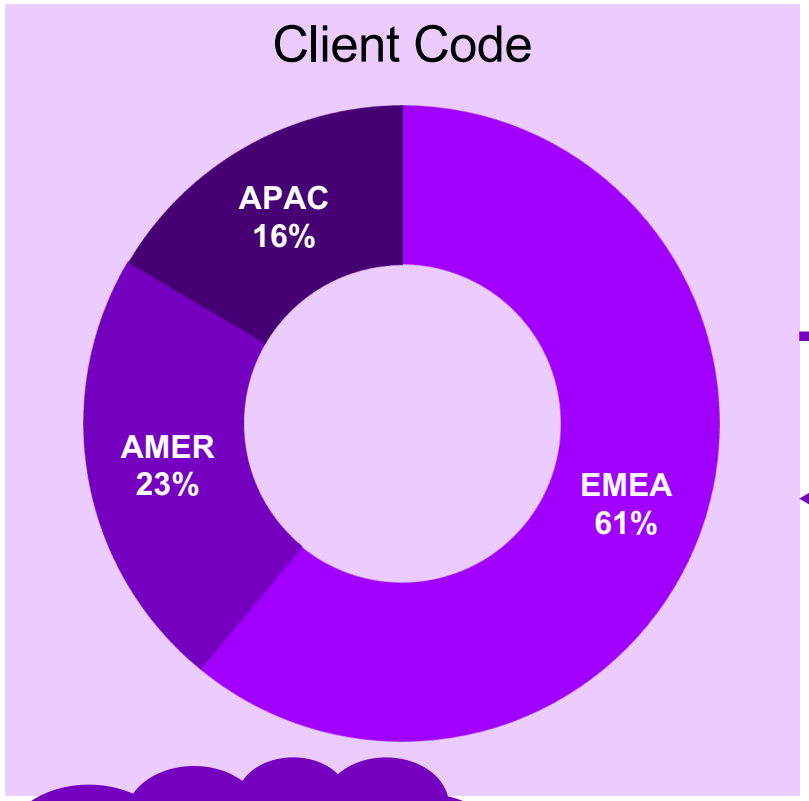
Why Do We Need Code in the Database?

Processing Data



Bring Code to the Data

Minimize communication for good performance



One Request

One Response Packet

Database (some TB)

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)

```plsql
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;
/
```

Static SQL

Compile errors & no SQLi

Usages in the data dictionary

PL/SQL – Without Data Processing (DML)?

```plsql
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 wrapper 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
11.0.20 in 23.3

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

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

```
declare
   co_user constant all_users.username%type := 'DEMO';
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 (23c),
mle-encode-base64 (23c)

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 wrapper 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

```
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

```
set define off
script
var url = new java.net.URL("https://esm.run/validator@13.11.0");
var content = new java.lang.String(url.openStream().readAllBytes(),
                java.nio.charset.StandardCharsets.UTF_8);
var script = "create or replace mle module validator_mod "
                + "language javascript as " + "\n"
                + content + "\n"
                + "/" + "\n";
sqlcl.setStmt(script);
sqlcl.run();
/
```

Requires SQLcl or SQL Developer

Disconnected from npm

Single object in the Oracle data dictionary

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

Custom SQLcl Command

JavaScript – MLE Module Wrapper

```
create or replace package validator_api is
   function is_email(
      in_email in varchar2
   ) return boolean as mle module validator_mod signature 'default.isEmail(string)';

   function is_email(
      in_email   in varchar2,
      in_options in json
   ) return boolean as mle module validator_mod signature 'default.isEmail(string, any)';

   function is_email_djs(
      in_email   in varchar2,
      in_options in json default null
   ) return boolean;
end validator_api;
/
```

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

```
create or replace package body validator_api is
   function is_email_djs(
      in_email   in varchar2,
      in_options in json default null
   ) return boolean is
      co_js    constant clob := q'~
         (async () => {
            const bindings = await import("mle-js-bindings");
            const email = bindings.importValue("email");
            const options = bindings.importValue("options");
            const validator = await import("validator");
            const result = validator.default.isEmail(email, options);
            bindings.exportValue("result", result);
         })();
      ~';
      l_ctx    dbms_mle.context_handle_t;
      l_options json;
      l_result boolean;
   begin
      l_options := coalesce(in_options, JSON('{}'));
      l_ctx := dbms_mle.create_context(environment => 'DEMO_ENV');
      dbms_mle.export_to_mle(l_ctx, 'email', in_email);
      dbms_mle.export_to_mle(l_ctx, 'options', l_options);
      dbms_mle.eval(l_ctx, 'JAVASCRIPT', co_js);
      dbms_mle.import_from_mle(l_ctx, 'result', l_result);
      dbms_mle.drop_context(l_ctx);
      return l_result;
   end is_email_djs;
end validator_api;
/
```

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

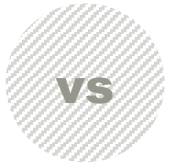
Can handle different module versions

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

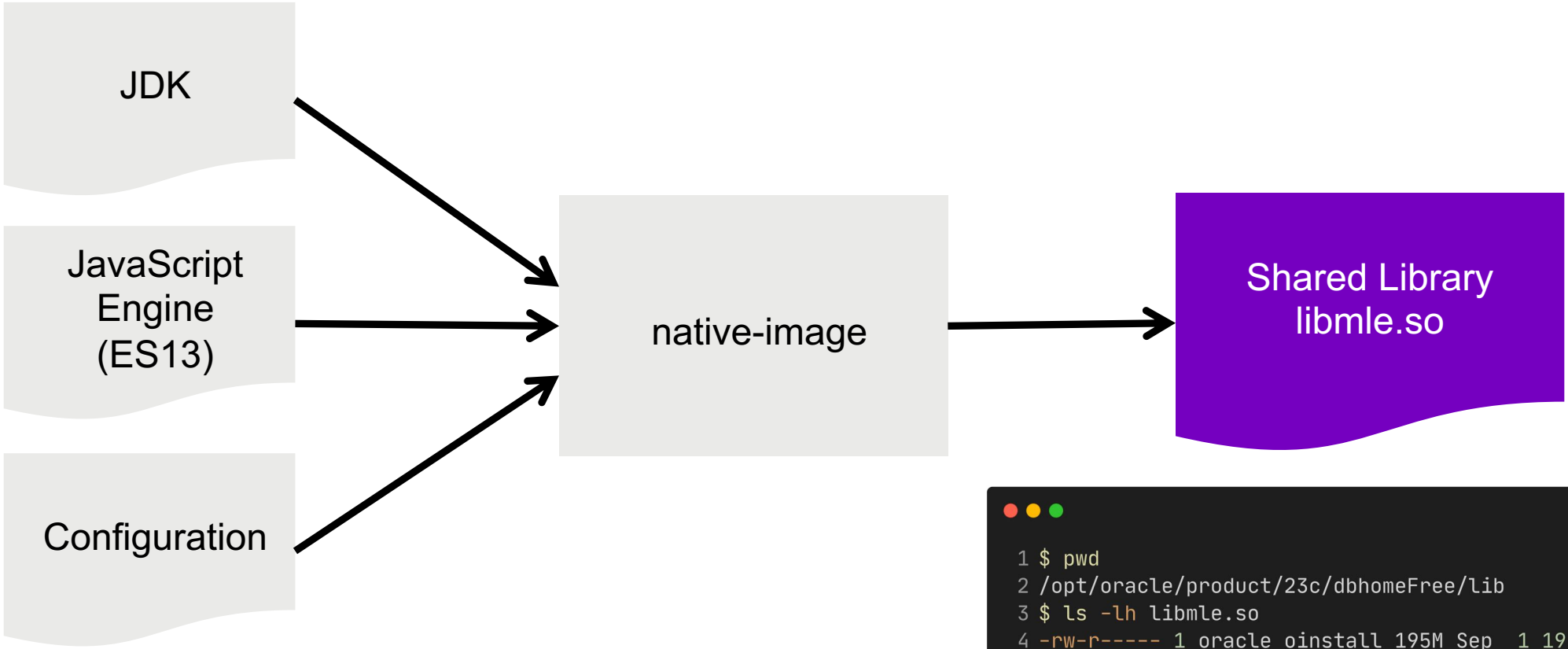
How Many JVMs are in the Oracle Database?

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



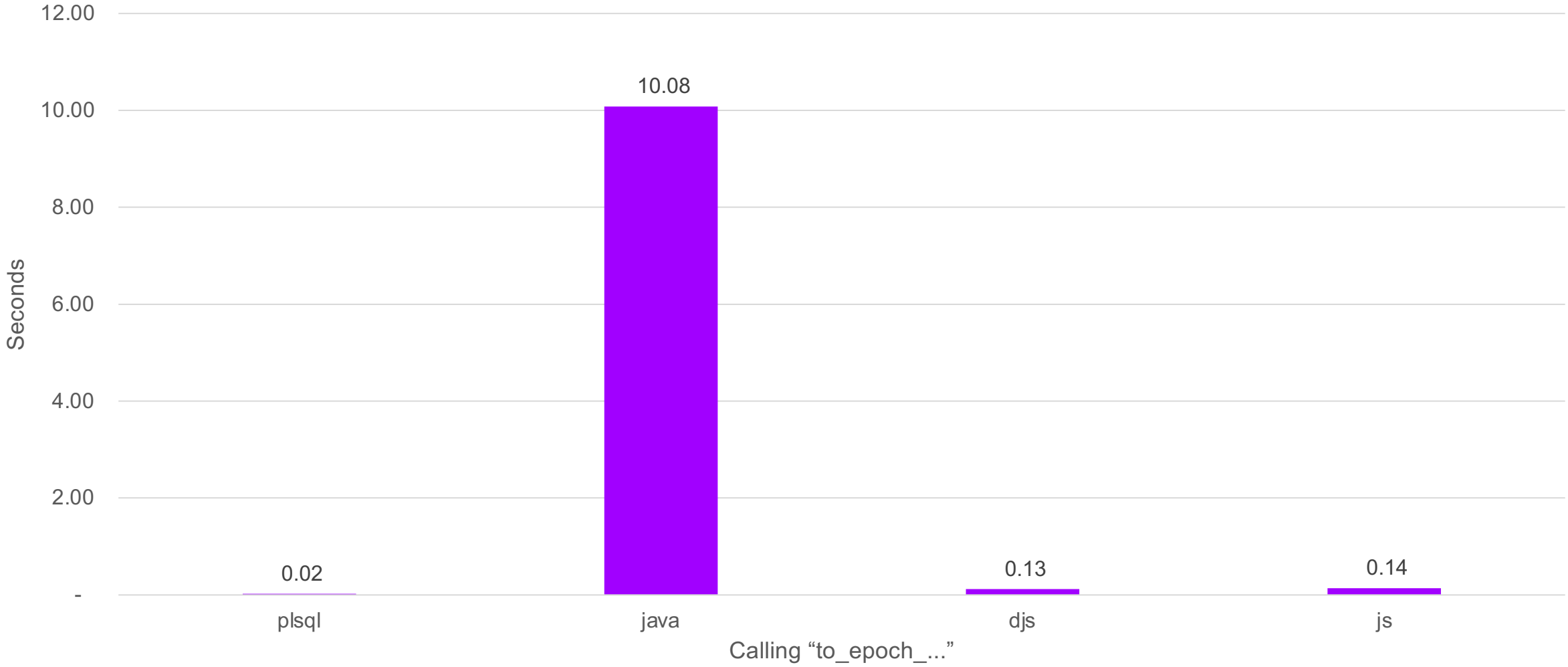

```
$ pwd
/opt/oracle/product/23c/dbhomeFree/lib
$ ls -lh libmle.so
-rw-r----- 1 oracle oinstall 195M Sep  1 19:00 libmle.so
```

Comparing Performance & Resource Usage

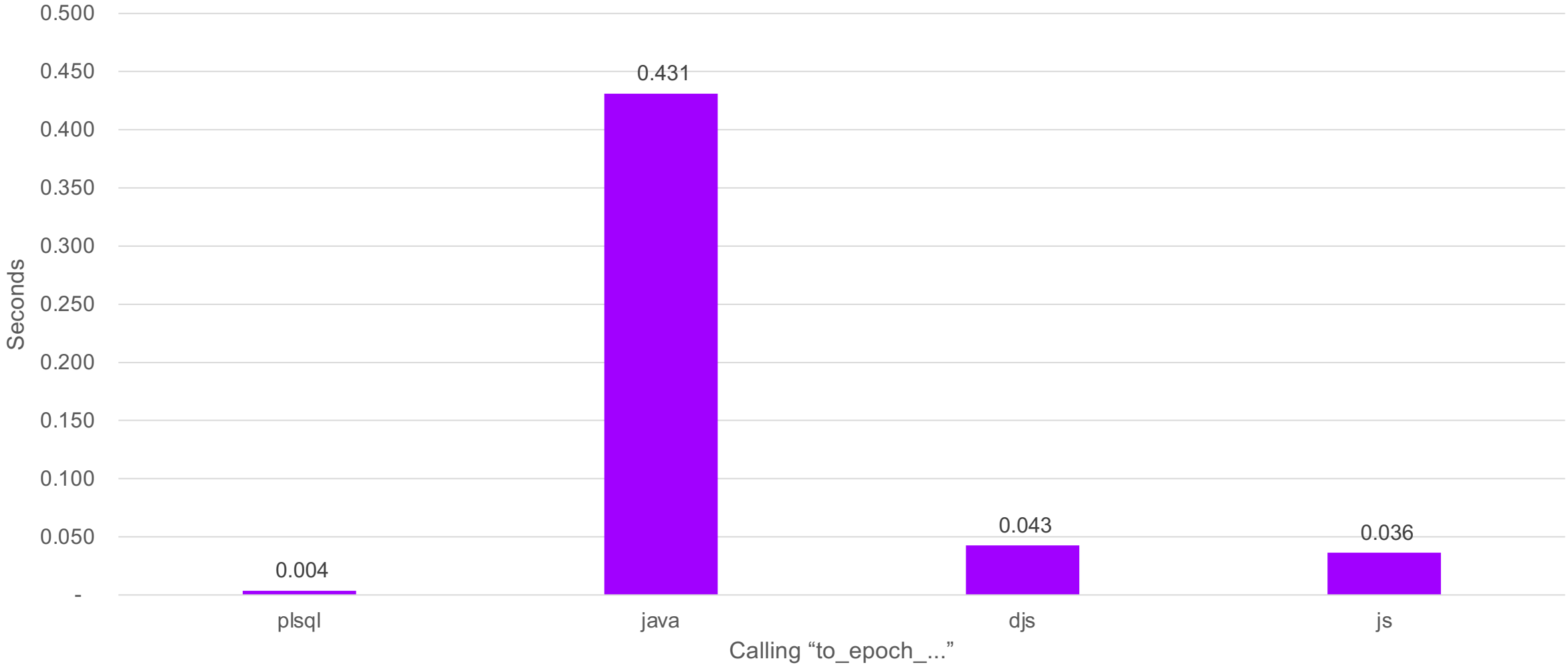
Best of 3 Attempts

>

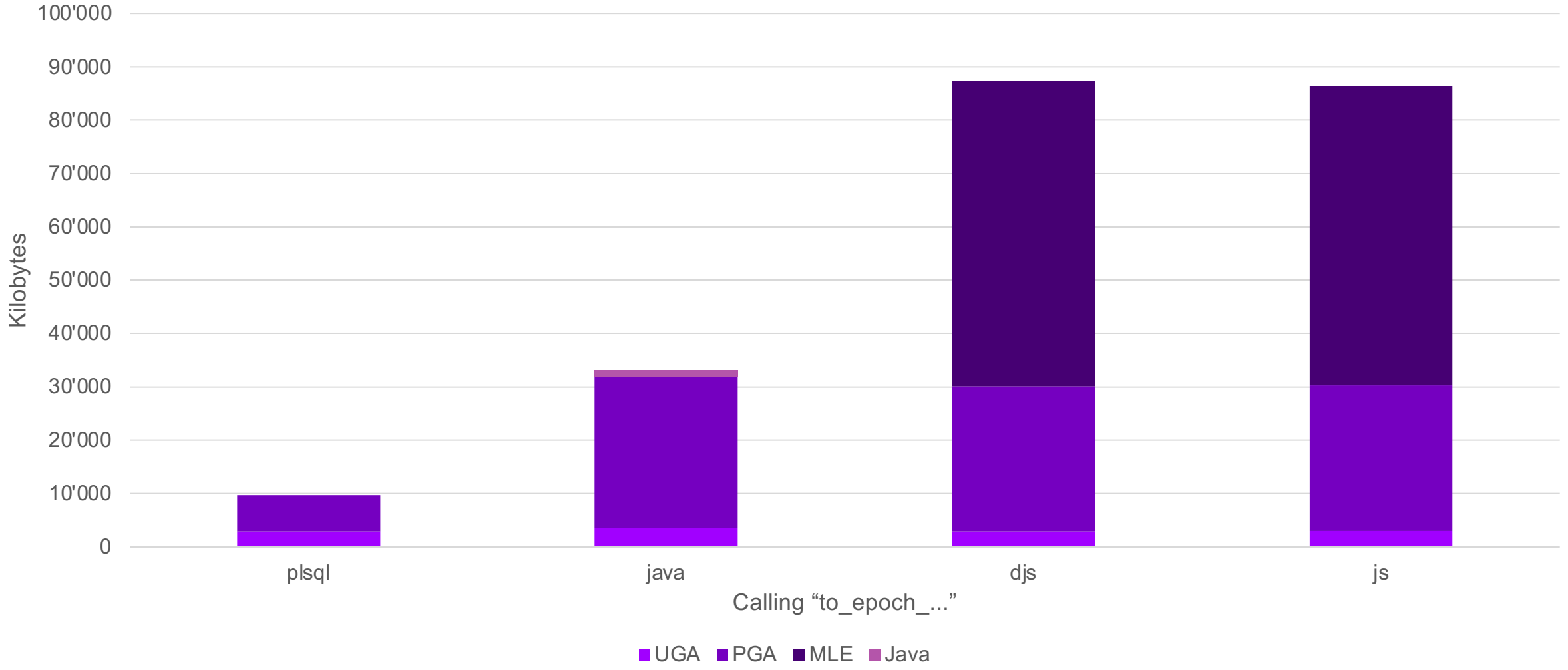
Runtime of First Call after DB Restart



Runtime of First Call in New DB Session

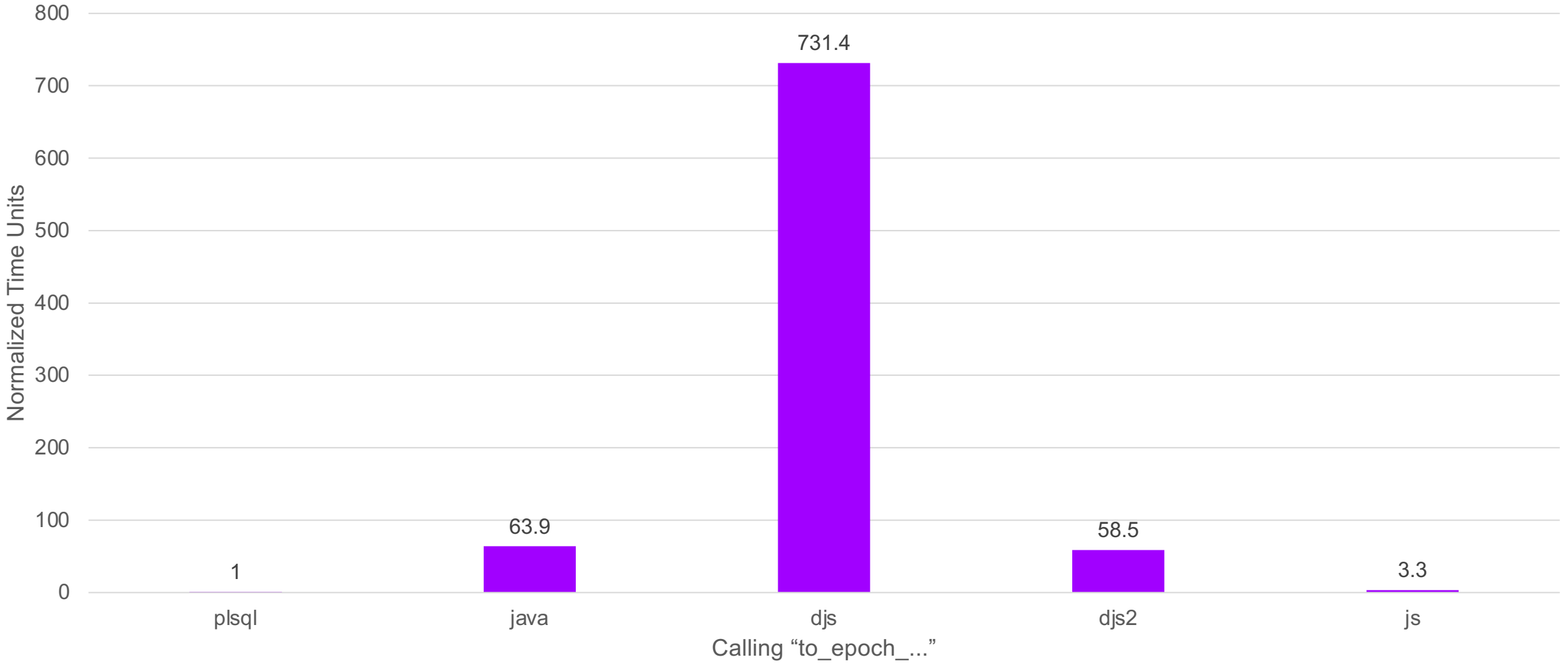


Max. Memory Usage After Single Call

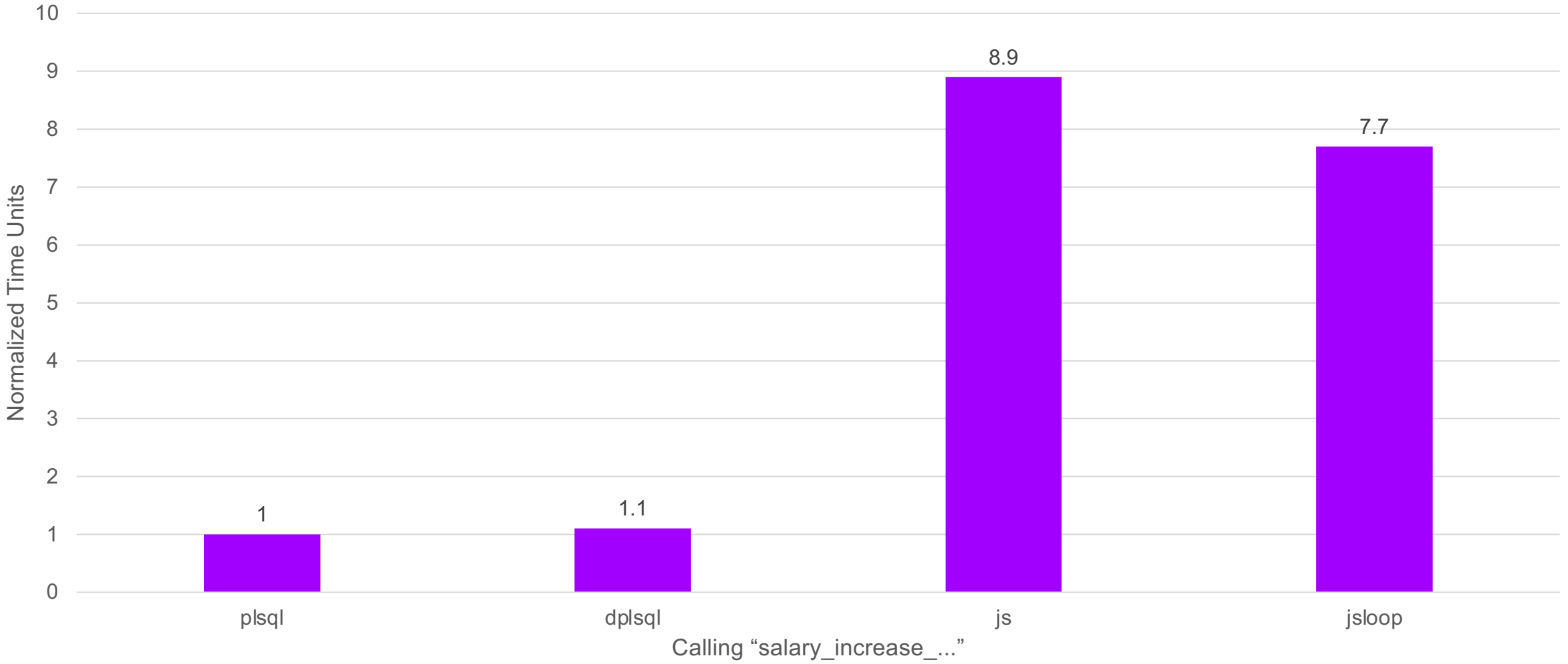


Comparing Apples with Pears?

Runtime of 100’000 Calls of to_epoch



Runtime of 100’000 Calls of salary_increase



Good Security Practices & Default Exceptions

Binds and Assertions

```
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)`
  );
}
/
```

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

Template literals and
JSON integration are great

Source: based on https://docs.oracle.com/en/database/oracle/oracle-database/23/mlejs/mle-security.html (Example 9-3)

Assert in JavaScript

```
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)`
  );
}
/
```

```
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 (DEMO.SQL_ASSERT_MOD:7:463)
ORA-06512: at "DEMO.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/
```

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

Get Missing Error Stack

```
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 (DEMO.SQL_ASSERT_MOD:7:463)
ORA-06512: at "DEMO.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/
```

"You cannot be serious!"

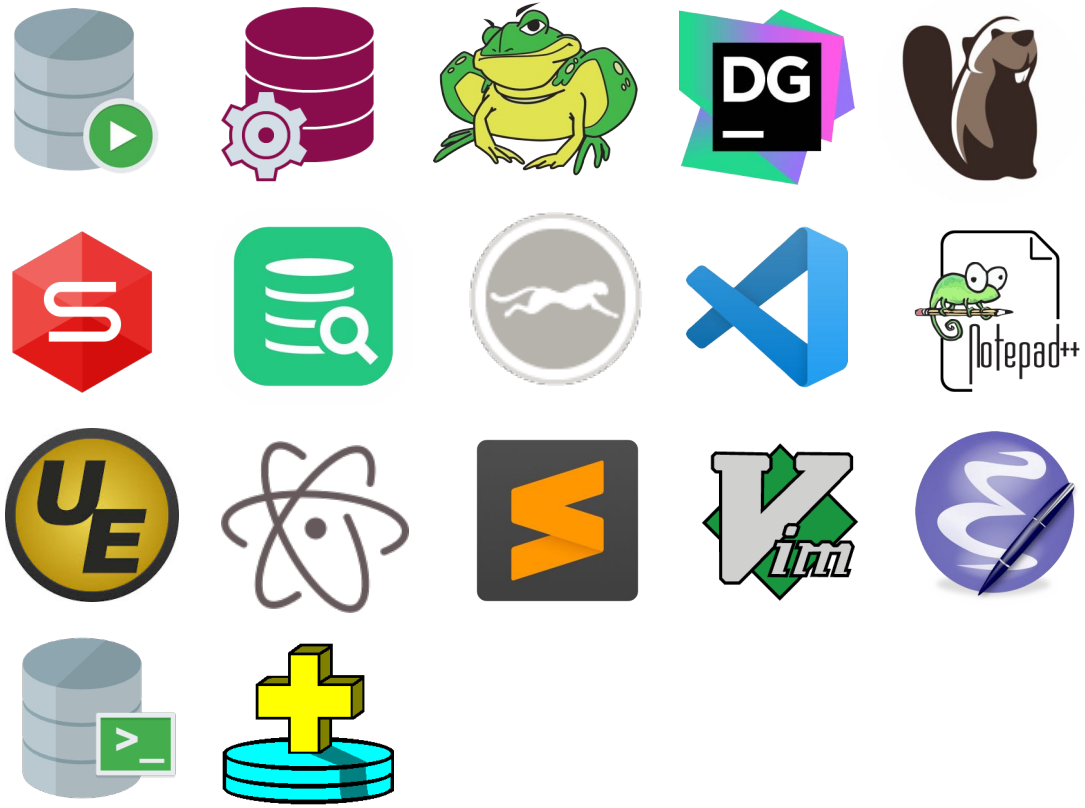
John McEnroe, June 22, 1981

```
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 (DEMO.CREATE_TEMP_TABLE_MOD:6:20)


PL/SQL procedure successfully completed.
```

Development & Tooling

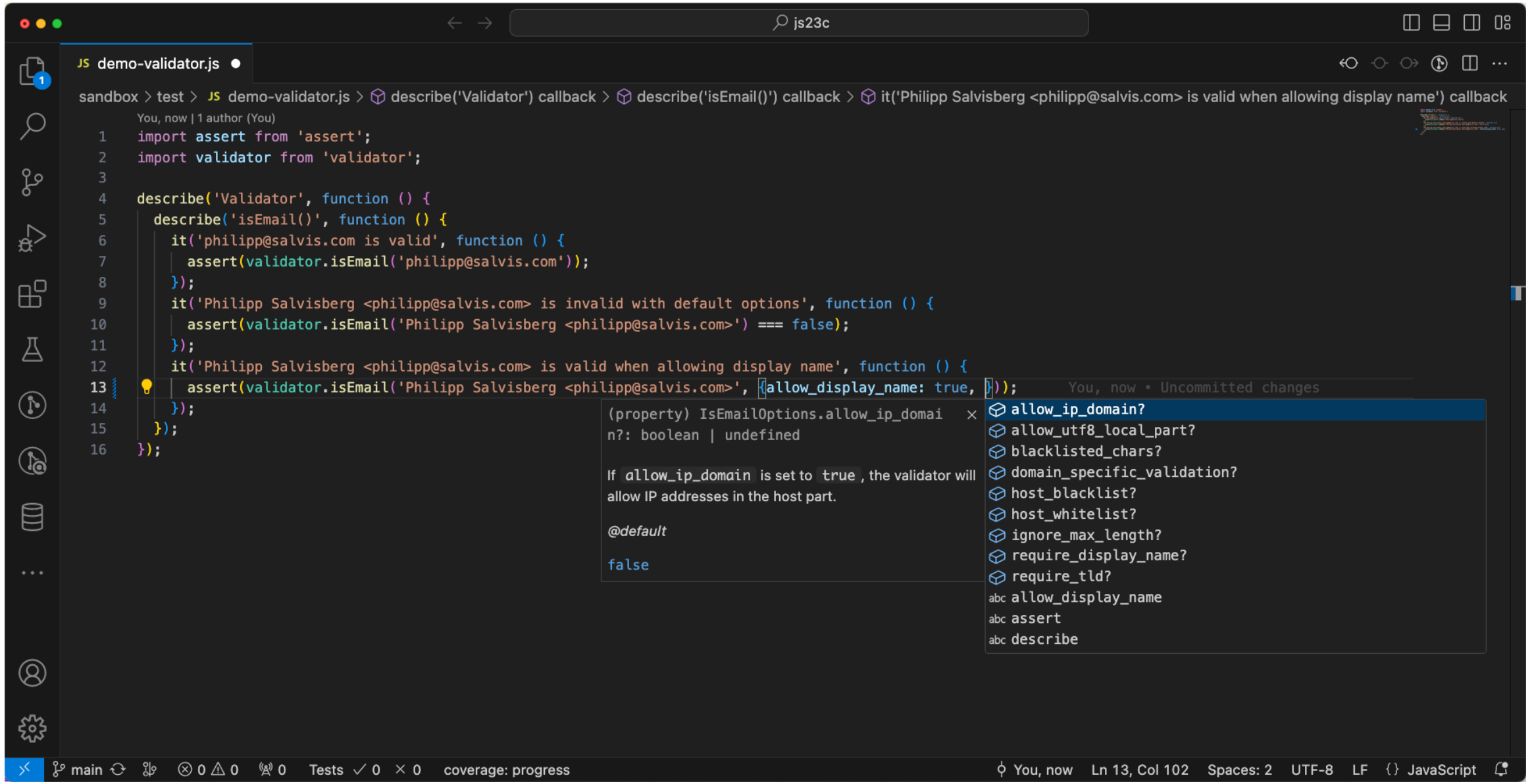
Development Tools

SQL & PL/SQL

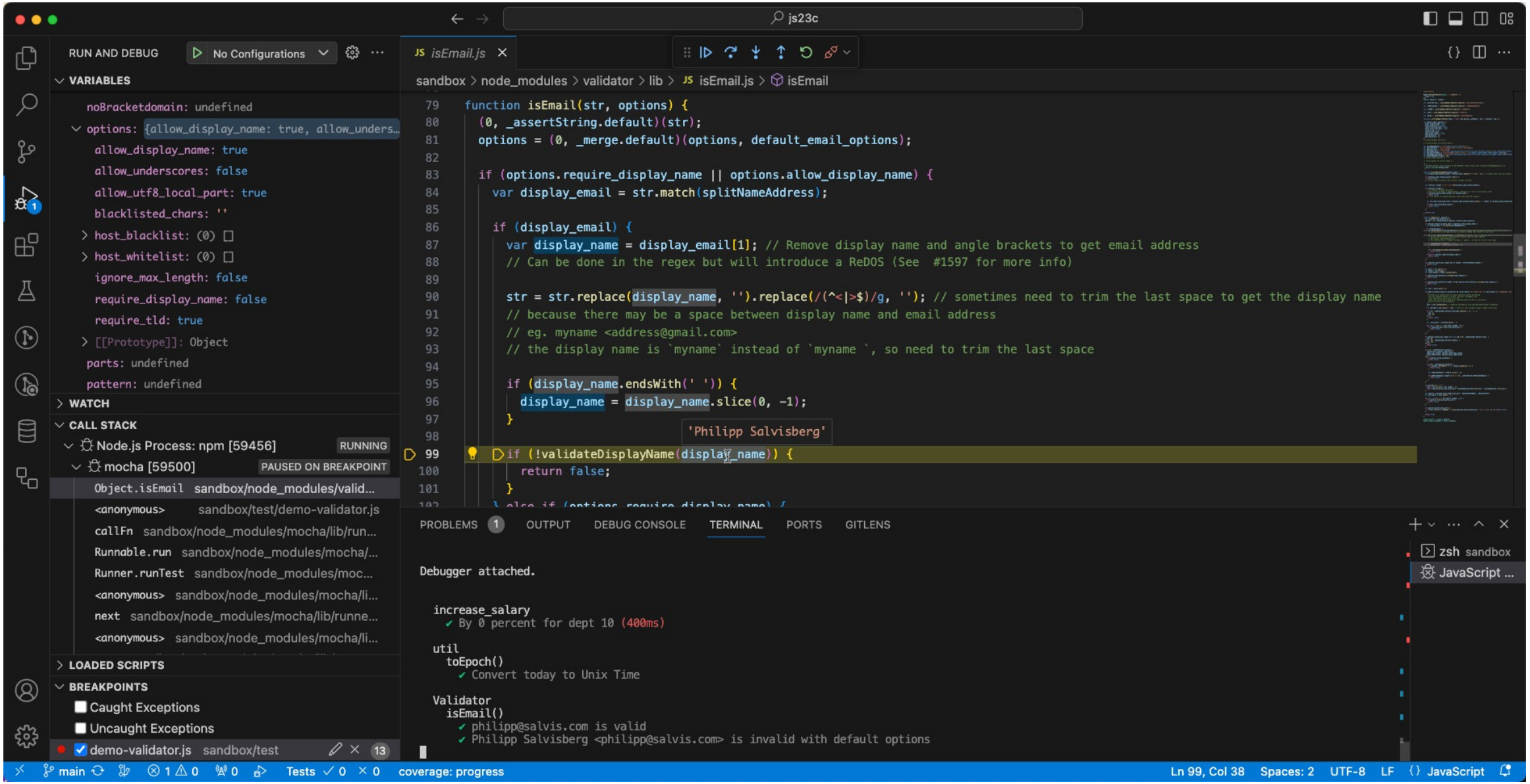


JavaScript

IntelliSense in VS Code



Debugging in VS Code

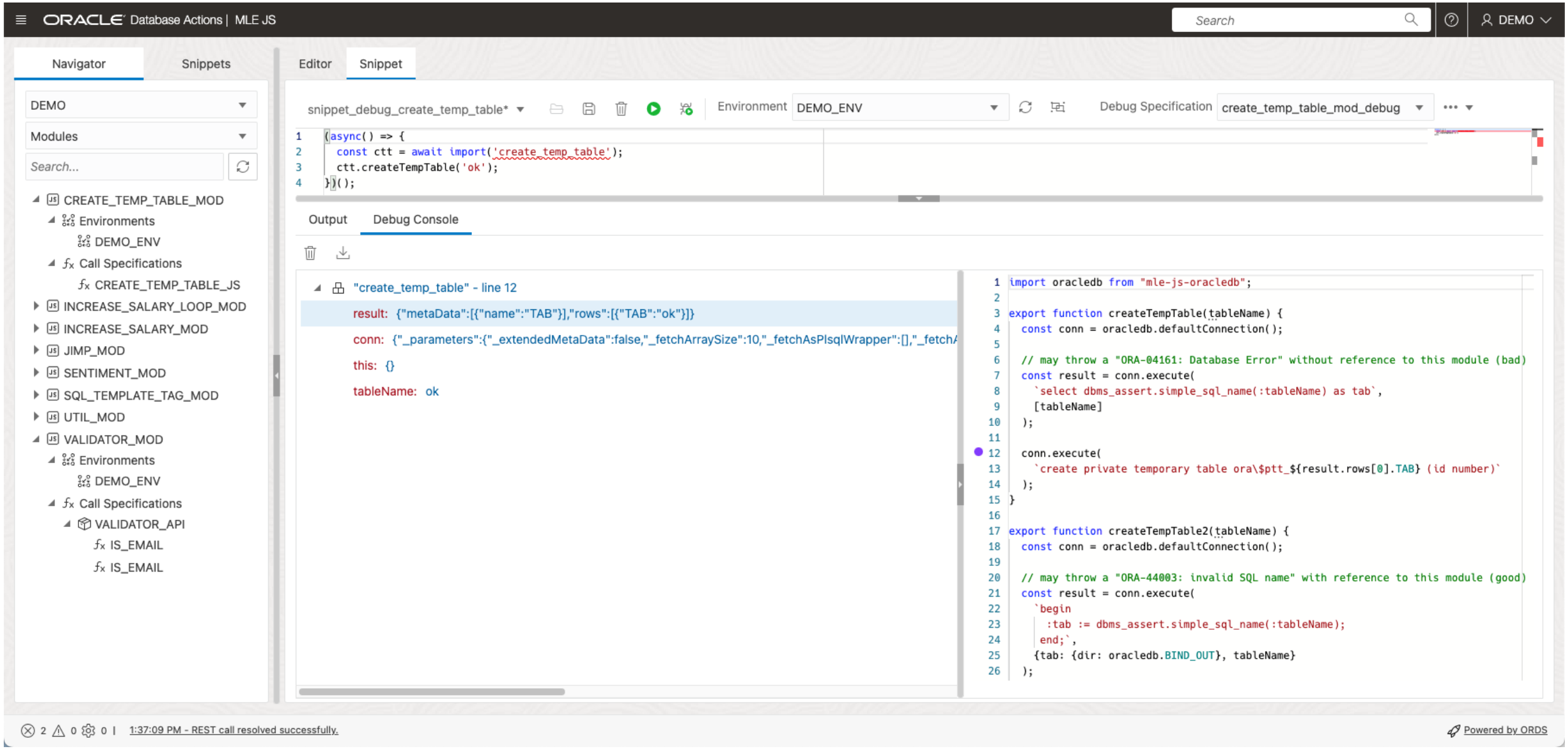


MLE Post-Execution Debugging

```
set serveroutput on size unlimited
declare
   co_breakpoints constant json := json(q'~
{
  "version": "1.0",
  "debugpoints": [
    {
      "at": {"name": "CREATE_TEMP_TABLE_MOD", "line": 4},
      "actions": [{"type": "watch", "id": "tableName"}],
      "condition": "tableName.includes('-')"
    },
    {
      "at": {"name": "CREATE_TEMP_TABLE_MOD", "line": 12},
      "actions": [{"type": "snapshot"}]
    }
  ]
}
~');
   l_sink         blob;
   l_output       json;
begin
   dbms_lob.createtemporary(l_sink, false, dbms_lob.call);
   dbms_mle.enable_debugging(co_breakpoints, l_sink);
   ut.run('test_create_temp_table.create_invalid_temp_table_js');
   l_output := dbms_mle.parse_debug_output(l_sink);
   dbms_output.put_line('MLE debug output: '
      || chr(10)
      || json_serialize(l_output returning clob pretty));
   dbms_mle.disable_debugging();
end;
/
```

```
all
  test_create_temp_table
    js
      create_invalid_temp_table_js [.039 sec]

Finished in .043893 seconds
1 tests, 0 failed, 0 errored, 0 disabled, 0 warning(s)

MLE debug output:
[
  [
    {
      "at" :
      {
        "name" : "DEMO.CREATE_TEMP_TABLE_MOD",
        "line" : 4
      },
      "values" :
      {
        "tableName" : "TEST-JS"
      }
    }
  ]
]


PL/SQL procedure successfully completed.
```

Debugging in Database Actions



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

vs

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:

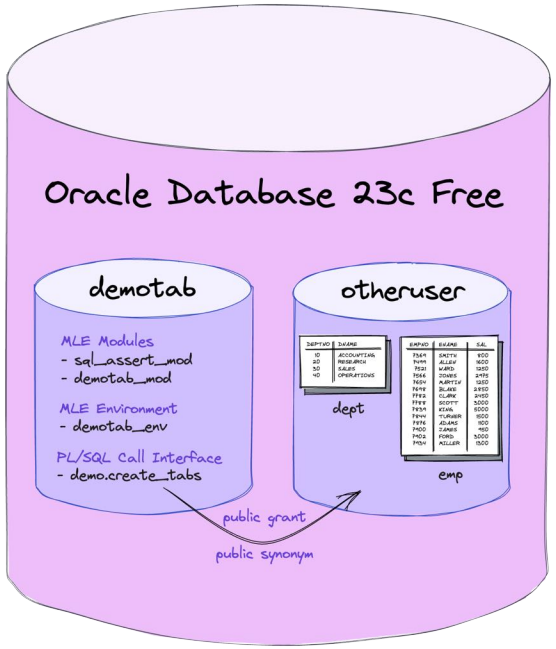
- You should do it in a single SQL statement if at all possible. And believe it or not, it is almost always possible. This statement is even truer as time goes on. SQL is an extremely powerful language.
- If you can't do it in a single SQL Statement, do it in PL/SQL—as little PL/SQL as possible! Follow 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 Oracle9*i* 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