Writing fast trusted stored functions in PL/Rust

Jim Mlodgenski

Senior Principal Engineer
Amazon Web Services

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Procedural Language Overview

PostgreSQL allows user-defined functions to be written in a variety of procedural languages

The database server has no built-in knowledge about how to interpret or execute the function’s source text

PostgreSQL has many procedural languages

- PL/pgSQL
- PL/Tcl
- PL/Perl
- PL/Python
- plv8
- PL/Rust
- And more…

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is a trusted procedural language?

“Trusted” is no indication of quality, it is just an indicator of the potential access a function may have

Trusted languages do not allow access to database server internals, the file system and other resources like the network

Only superusers can create functions with untrusted languages

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is Rust?

Developed by Mozilla as a language to give the performance of a low level language like C with things like memory management of higher level languages

Open sourced under the Apache and MIT licenses

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

What is PL/Rust?

Allows writing PostgreSQL functions in Rust

Uses the pgrx extension to bridge Rust code to PostgreSQL internals

Can be configured as a trusted language so non-superusers can write functions with it

```
CREATE FUNCTION one()
  RETURNS int
AS
$$
    Ok(Some(1))
$$ LANGUAGE plrust;
```

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Creating a PL/Rust function

```
CREATE FUNCTION one()
  RETURNS int
AS
$$
    Ok(Some(1))
$$ LANGUAGE plrust;
```

- Write to pg_proc
- PL/Rust Validator
  - Write lib.rs and crate.toml
  - Check lints
  - Compile
  - Update pg_proc

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Storing a compiled function

```
--
-- Name: one(); Type: FUNCTION; Schema: public; Owner: postgres
--

CREATE FUNCTION public.one() RETURNS integer    LANGUAGE plrust    AS
$${"src":"\n    Ok(Some(1))\n","trusted_pgrx_version":"=1.2.7","lib":{"aarch64-
postgres-linux-
gnu":{"encoding":"GzBase64","symbol":"plrust_fn_oid_16695_16702","encoded":"H4sI
AAAAAAAC_-
39C3xU1dU3jp8zMyGTScgFAoRrJtwMAcLcL4oy4SIBEWKIotZ2MplLEsmNZKLgpQRFBaGVKK0-
aBXUtiRKSytWbW0N2guttvVWS_v4tAHt0yBYCUUlCuT_XXvvM3PmMMOl9Xl-7_v5v4GZffY6a6-
99tprrb323ufsWbtgyeU6WZaUP730nES56lE87xNw33_EcXySR0rH9xRpMsNNk1L_7X82MZUkOfY9BJ8
Tz3HoiefkhLTAyOHVxsRyOlHO8iKHWl6UE1KzTkpIleLp-
BjOUp_ylycllqv4WzRE5dY_wvNnpIbEVCTSVSg3RDr_v1yRVor6UvFpFIwpqcL9fHyKxXUmPjkqXsYmq
Y_kMQWfSSrYVHxG4ENdn5-kTJYmPwafcfgUqWCF-IyP6RL6AZ-
LktAaic9klcyz8TEl0SVjkrITRTpMpKNV9yaorodK__5fxgXgFqiuhyfp2wvRBy4_o-
QZkQyee4be7pXj8lD_bUsB_6uKL_Xf31PAh-iS0-lIgb8oRb2_S4F_uapf1X_3p6BjkJPT-
SAF_tVMX0zS0bGJujw5BZ2lKej0p-D_pyngd6ag87UU-F8RfJ4UfL4v4LYUfO5IQT-cAn5bCnh5Cvo
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Why store the binary in pg_proc?

Performance - Compiling a PL/Rust function may take minutes so storing the binary avoids the first start penalty

Replication - Replicas need the binaries in a transactionally consistent way and cannot impose a first start penalty

Backups - pg_restore cannot take an extended period of time restoring many PL/Rust functions

Consistency - There are race conditions on REPLACE when considering a separate table in the plrust schema

aws

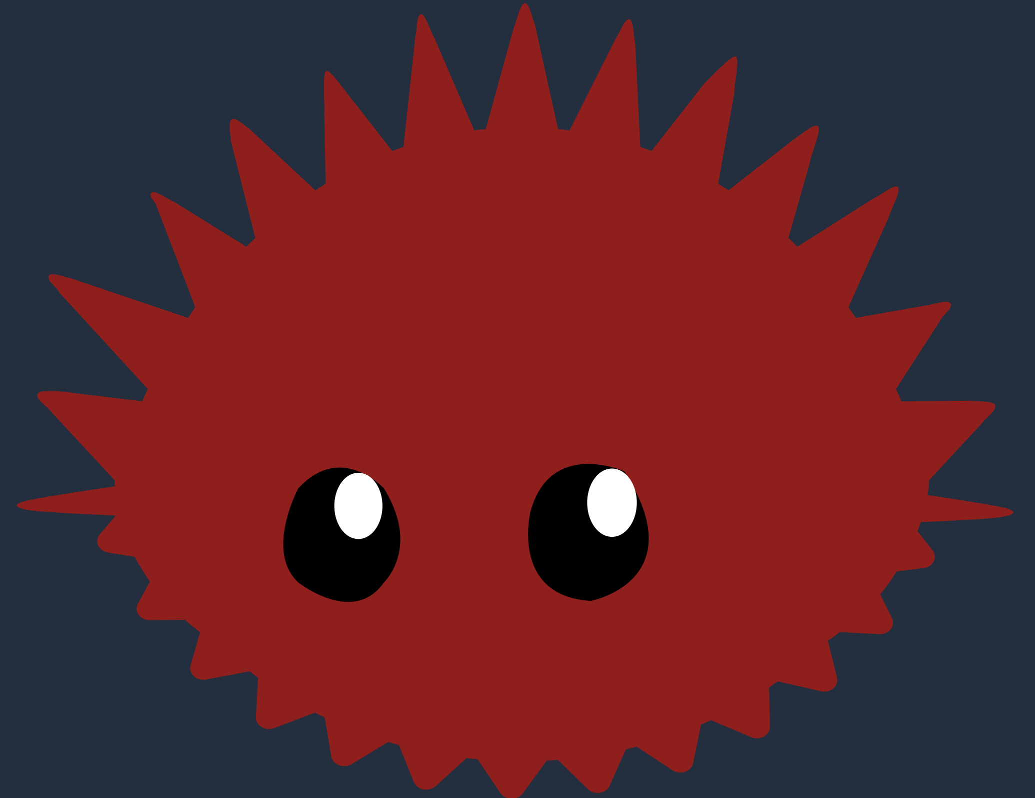
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Making PL/Rust trusted

Rust has some strong safety properties over C/C++ around memory

Prevents the use of “unsafe” Rust and other known safety issues

Uses a specialized standard library to control OS access



aws © 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Lints

Statically analyzes the user code to look for patterns that an known to violate safety

Throws errors at compile time

Uses a combination of standard Rust lints and custom PL/Rust lints

```
CREATE FUNCTION read_file()
  RETURNS text AS
$$
  let s = include_str!("/etc/passwd");
  Ok(Some(s.into()))
$$ LANGUAGE plrust;


ERROR:
   0: `cargo build` failed
...
   error: the `include_str`, `include_bytes`,
and `include` macros are forbidden in PL/Rust
     --> src/lib.rs:44:17
      |
   44 |         let s =
include_str!("/etc/passwd");
      |
^^^^^^^^^^^^^^^^^^^^^^^^^^^
      |
   note: the lint level is defined here
     --> src/lib.rs:25:15
```

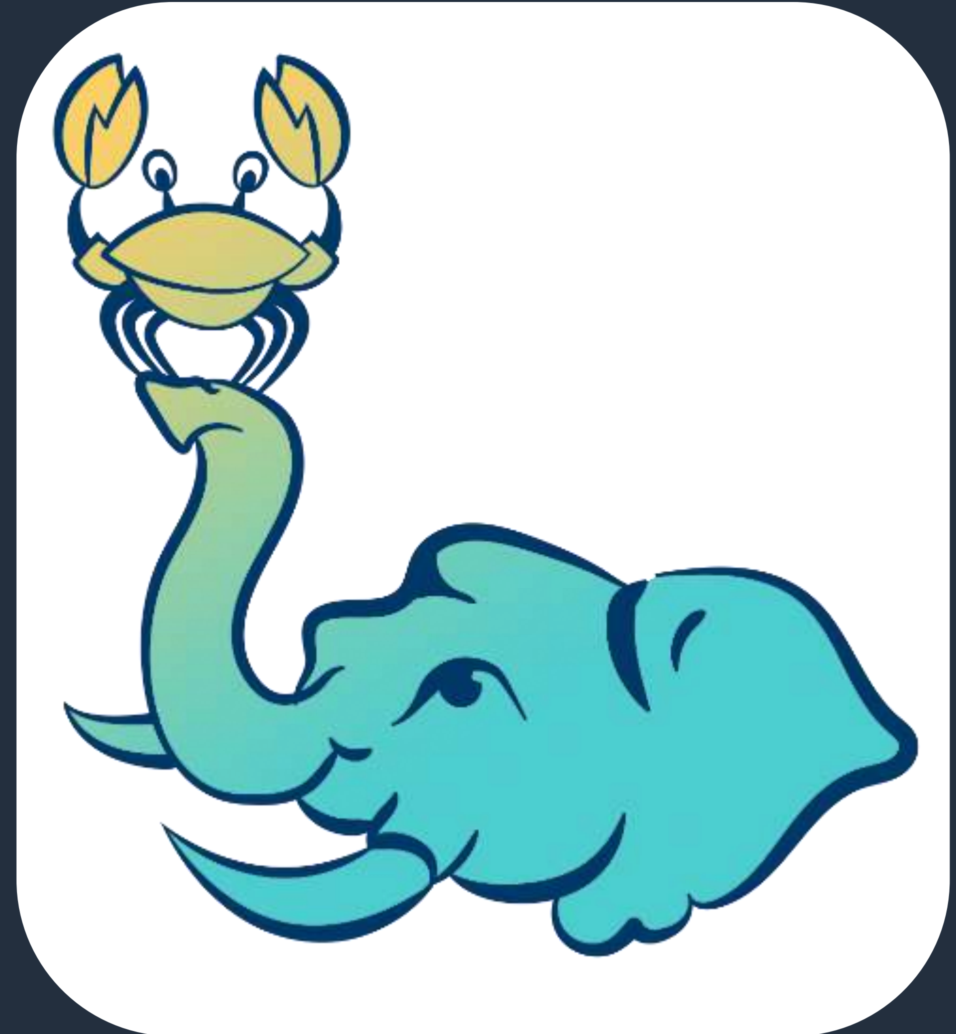
aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

postgrestd

Fork of the Rust standard library and implements a variant of std::sys for system bindings

Some functions are stubbed out preventing them from being used

Others are implemented using PostgreSQL features such as the global allocator using palloc



© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Configuring PL/Rust

PL/Rust has parameters to configure how functions are built

Allows PL/Rust to use an existing Rust environment if necessary

The level of “trustness” can be adjusted based on specific needs

plrust.work_dir

plrust.path_override

plrust.allowed_dependencies

plrust.compilation_targets

plrust.compile_lints

plrust.tracing_level

aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

PL/Rust structure

```
CREATE FUNCTION {fn_name} ({args})
RETURNS {ret}
-- function attributes can go here
AS $$
    // PL/Rust function body goes here
    // All PL/Rust functions return Result<Option<{ret}>>
$$ LANGUAGE plrust;
```

The return of a PL/Rust function needs to be of the format Result<Option<T>>

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Parameter strictness

The STRICT keyword determines if a function will execute on NULL input

PL/Rust uses STRICT to determine how a parameter is passed to the function

```
CREATE FUNCTION echo(input int)
  RETURNS int
AS
$$
  Ok(Some(input))
$$ STRICT
LANGUAGE plrust;

CREATE FUNCTION echo(input int)
  RETURNS int
AS
$$
  Ok(Some(input.unwrap_or_default()))
$$ LANGUAGE plrust;
```

aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Data types

Some map to native Rust types

PostgreSQL Type	Rust Type
bytea	Vec<u8> or &[u8]
text	String or &str
varchar	String or &str
smallint	i16
integer	i32
bigint	i64
oid	u32
real	f32
double precision	f64
bool	bool

Others map to PGRX types

PostgreSQL Type	Rust Type
json	Json(serde_json::Value)
jsonb	JsonB(serde_json::Value)
date	Date
time	Time
timestamp	Timestamp
time with time zone	TimeWithTimeZone
timestamp with time zone	TimestampWithTimeZone
numeric	AnyNumeric
uuid	Uuid([u8; 16])
int4range	Range<i32>

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Data types

```
CREATE FUNCTION sum_and_double (a numeric, b numeric)
  RETURNS numeric AS
$$
  Ok(Some((a + b) * 2))
$$ STRICT
LANGUAGE plrust;


CREATE FUNCTION moon_landing()
  RETURNS date AS
$$
  Ok(Some(Date::new(1969, 7, 20)?))
$$ STRICT
LANGUAGE plrust;
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Composite types

Uses PgHeapTuple as a structure with getter and setter methods

```
CREATE FUNCTION get_coords(x int, y int, z int)
  RETURNS coords AS
$$
  let mut c = PgHeapTuple::new_composite_type("coords")?;

  c.set_by_name("x", x)?;
  c.set_by_name("y", y)?;
  c.set_by_name("z", z)?;

  Ok(Some(c))
$$ STRICT
LANGUAGE plrust;
```

aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Running Queries

PL/Rust accesses database objects through SPI

There are a number of functions exposed to simplify SPI access

```
CREATE FUNCTION is_plrust_trusted()
  RETURNS bool AS
$$
  Ok(Spi::get_one("SELECT lanpltrusted FROM
  pg_language WHERE lanname = 'plrust'")?)
$$ LANGUAGE plrust;
```

aws © 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Updating rows

SPI is fully available to do more complex things like changing rows or prepared statements

```
CREATE FUNCTION add_foo(input int)
  RETURNS void AS
$$
  Spi::connect(|mut client| {
    client.update(
      "INSERT INTO foo VALUES ($1)",
      None, Some(vec![(PgBuiltInOids::INT4OID.oid(),
                  input.into_datum())])).map(|_| ())
  })?;

  Ok(Some(()))
$$ STRICT
LANGUAGE plrust;
```

aws

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Logging

PL/Rust uses the PostgreSQL logging infrastructure

It provides several macros for simplicity

Full ereport functionality is available when needed

```
CREATE OR REPLACE FUNCTION one()
  RETURNS int
AS
$$
  log!("This is a log message");
  notice!("This is a notice");
  warning!("This is a warning");

  ereport!(PgLogLevel::LOG,
 PgSqlErrorCode::ERRCODE_SUCCESSFUL_COMPLETION,
    "Full ereport is available");

  Ok(Some(1))
$$ LANGUAGE plrust;
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Dynamic function calling

Allows for calling other functions directly without going through SPI

Exposes the PostgreSQL function call interface to PL/Rust

Can call any function type including inbuilt, extensions and user defined from other languages

```
CREATE FUNCTION rsoundex(w text)
  RETURNS text AS
$$
  let result = fn_call("soundex",
                 &[&Arg::Value(w)])?;

  Ok(result)
$$ STRICT LANGUAGE plrust;
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Triggers

```
CREATE FUNCTION double_foo_trigger()
  RETURNS trigger AS
$$
    let tg_op = trigger.op()?;

    let my_new = match tg_op {
        INSERT => trigger.new().unwrap(),
        _ => error!("This trigger only applies to inserts")
    };
    let mut my_new = my_new.into_owned();

    let col_name = "a";
    match my_new.get_by_name::<i32>(col_name)? {
        Some(val) => my_new.set_by_name(col_name, val * 2)?,
        None => (),
    }

    Ok(Some(my_new))
$$
LANGUAGE plrust;
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Crates

Crates are packages that add capabilities to Rust

Trusted PL/Rust has an allow list of crates that can be used

These can be listed by the function plrust.allowed_dependencies

```
CREATE FUNCTION randint()
  RETURNS bigint AS
$$
[dependencies]
rand = "0.8"

[code]
use rand::Rng;
Ok(Some(rand::thread_rng().gen()))
$$ LANGUAGE plrust;
```

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Performance

Baseline plpgsql function inspired from HammerDB
https://github.com/TPC-Council/HammerDB

```
CREATE OR REPLACE FUNCTION dbms_random(start_int int, end_int int)
  RETURNS int AS
$$
BEGIN
  RETURN trunc(random() * (end_int-start_int + 1) + start_int);
END
$$ LANGUAGE plpgsql STRICT;
```

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Performance

```
postgres=> EXPLAIN ANALYZE SELECT dbms_random(1, g)
                   FROM generate_series(1, 1000000) g;
                      QUERY PLAN
-----------------------------------------------------------------
 Function Scan on generate_series g
    (cost=0.00..260000.00 rows=1000000 width=4)
    (actual time=53.588..590.151 rows=1000000 loops=1)
 Planning Time: 0.031 ms
 Execution Time: 620.852 ms
(3 rows)
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Performance

PL/Rust using SPI

```
CREATE FUNCTION dbms_random (start_int int, end_int int)
  RETURNS int AS
$$
  Ok(Spi::get_one_with_args(
                  "SELECT trunc(random() * ($1-$2 + 1) + $3)::int",
                  vec![(PgBuiltInOids::INT4OID.oid(),
                       end_int.into_datum()),
                      (PgBuiltInOids::INT4OID.oid(),
                       start_int.into_datum()),
                      (PgBuiltInOids::INT4OID.oid(),
                       start_int.into_datum())],
              )?)
$$ LANGUAGE plrust STRICT;
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Performance

```
postgres=> EXPLAIN ANALYZE SELECT dbms_random(1, g)
                   FROM generate_series(1, 1000000) g;
                      QUERY PLAN
------------------------------------------------------------------
 Function Scan on generate_series g
    (cost=0.00..260000.00 rows=1000000 width=4)
    (actual time=55.461..14142.495 rows=1000000 loops=1)
 Planning Time: 0.049 ms
 Execution Time: 14191.605 ms
(3 rows)
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Performance

PL/Rust using a dynamic function call

```
CREATE FUNCTION dbms_random (start_int int, end_int int)
  RETURNS int AS
$$
  let r = fn_call::<f64>("random", &[])?;

  Ok(Some(f64::trunc((r.unwrap() *
                     ((end_int - start_int + 1) as f64))
                            + start_int as f64) as i32))
$$ LANGUAGE plrust STRICT;
```

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Performance

```
postgres=> EXPLAIN ANALYZE SELECT dbms_random(1, g)
                   FROM generate_series(1, 1000000) g;
                      QUERY PLAN
-----------------------------------------------------------------
 Function Scan on generate_series g
    (cost=0.00..260000.00 rows=1000000 width=4)
    (actual time=55.750..1851.216 rows=1000000 loops=1)
 Planning Time: 0.032 ms
 Execution Time: 1896.496 ms
(3 rows)
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Performance

PL/Rust using a native crate

```
CREATE FUNCTION DBMS_RANDOM (start_int int, end_int int)
  RETURNS int AS
$$
  [dependencies]
  rand = "0.8.5"

  [code]
  use rand::Rng;

  Ok(Some(rand::thread_rng().gen_range(start_int..(end_int + 1))))
$$ LANGUAGE plrust STRICT;
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Performance

```
postgres=> EXPLAIN ANALYZE SELECT dbms_random(1, g)
                   FROM generate_series(1, 1000000) g;
                      QUERY PLAN
-----------------------------------------------------------------
 Function Scan on generate_series g
    (cost=0.00..260000.00 rows=1000000 width=4)
    (actual time=55.754..309.444 rows=1000000 loops=1)
 Planning Time: 0.032 ms
 Execution Time: 339.965 ms
(3 rows)
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Better performance case

```
CREATE FUNCTION fib_plpgsql
                     (n numeric)
  RETURNS numeric AS
$$
DECLARE
  a numeric := 0;
  b numeric := 1;
  c numeric;
BEGIN
  IF n = 0 THEN
    RETURN a;
  END IF;
```

```
 FOR i IN 2..n LOOP
    c := a + b;
    a := b;
    b := c;
  END LOOP;

  RETURN b;
END
$$ IMMUTABLE
LANGUAGE plpgsql;
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Better performance case

```
CREATE FUNCTION fib_plrust
                  (n numeric)
  RETURNS numeric AS
$$
[dependencies]
num-bigint = "0.4.4"

[code]
  use num_bigint::BigUint;

  let mut a: BigUint = 0_u32.into();
  let mut b: BigUint = 1_u32.into();
  let ln: u32 =
           u32::try_from(n.unwrap())?;
```

```
  if ln <= 0 {
     return Ok(Some(
               AnyNumeric::try_from(
                   a.to_string()
                .as_str()).unwrap()));
   }

   for _ in 1..ln {
     let c = a + &b;
     a = b;
     b = c;
   }

   Ok(Some(AnyNumeric::try_from(
     b.to_string().as_str()).unwrap()))
$$ IMMUTABLE
LANGUAGE plrust;
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Better performance case

```
postgres=> SELECT fib_plpgsql(627177) = fib_plrust(627177);
 ?column?
----------
 t
(1 row)


postgres=> SELECT fib_plpgsql(627178);
ERROR:  value overflows numeric format
CONTEXT:  PL/pgSQL function fib_plpgsql(numeric) line 12 at
assignment
```

aws
© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Better performance case

```
postgres=> EXPLAIN ANALYZE SELECT fib_plpgsql(627177);
                         QUERY PLAN
-----------------------------------------------------------------
 Result  (cost=0.00..0.01 rows=1 width=32)
 (actual time=0.002..0.002 rows=1 loops=1)
 Planning Time: 61542.339 ms
 Execution Time: 0.018 ms
(3 rows)

Time: 61543.220 ms (01:01.543)
```

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Better performance case

```
postgres=> EXPLAIN ANALYZE SELECT fib_plrust(627177);
                                     QUERY PLAN
-----------------------------------------------------------------
 Result  (cost=0.00..0.01 rows=1 width=32)
 (actual time=0.002..0.002 rows=1 loops=1)
 Planning Time: 1181.729 ms
 Execution Time: 0.017 ms
(3 rows)

Time: 1182.533 ms (00:01.183)
```

aws © 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.

Thank you!

Jim Mlodgenski

© 2024, Amazon Web Services, Inc. or its affiliates. All rights reserved.