yugabyteDB

Using stored procedures effectively in a distributed PostgreSQL database

Bryn Llewellyn

Developer Advocate, Yugabyte Inc.

What is YugabyteDB?

YugaByte DB

Distributed SQL

PostgreSQL Compatible, 100% Open Source (Apache 2.0)

Massive Scale

Millions of IOPS in Throughput, TBs per Node

High Performance

Low Latency Queries

Cloud Native

Fault Tolerant, Multi-Cloud & Kubernetes Ready

Functional Architecture

YugaByte SQL (YSQL)

PostgreSQL-Compatible Distributed SQL API

DOCDB

Spanner-Inspired Distributed Document Store
Cloud Neutral: No Specialized Hardware Needed

Questions?

Download
download.yugabyte.com

Join Slack Discussions
yugabyte.com/slack

Star on GitHub
github.com/YugaByte/yugabyte-db

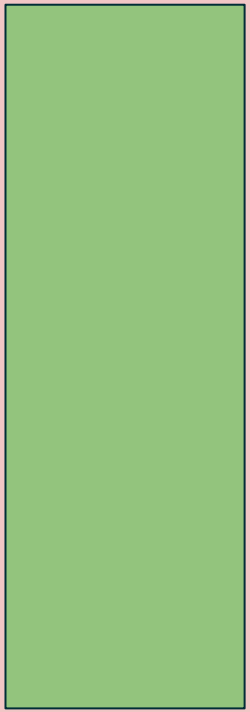
Q: Why use stored procedures?

- **Large software systems must be built from modules**
  - Hide implementation detail behind API
  - Software engineering's most famous principle
- **The RDBMS is a module**
  - Tables and SQLs that manipulate them are the implementation details
  - Stored procedures express the API
- **Result: happiness**
  - Developers and end-users of applications built this way are happy with their correctness, maintainability, security, and performance

A: Use stored procedures to encapsulate the RDBMS's functionality behind an impenetrable hard shell

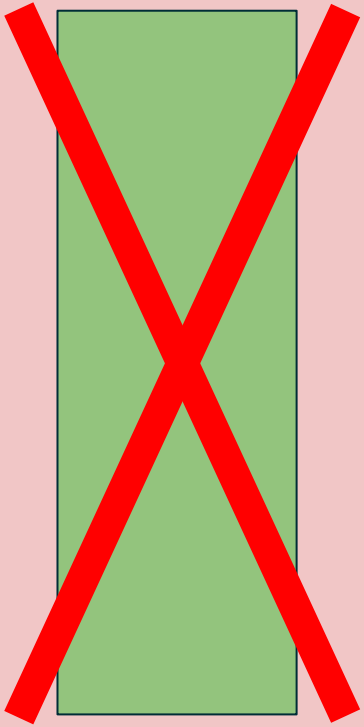
Hard Shell Schematic

Public



APP DATABASE

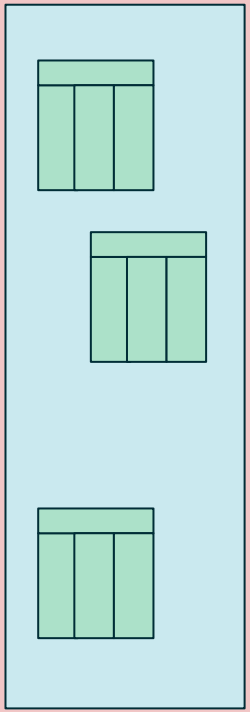
Public



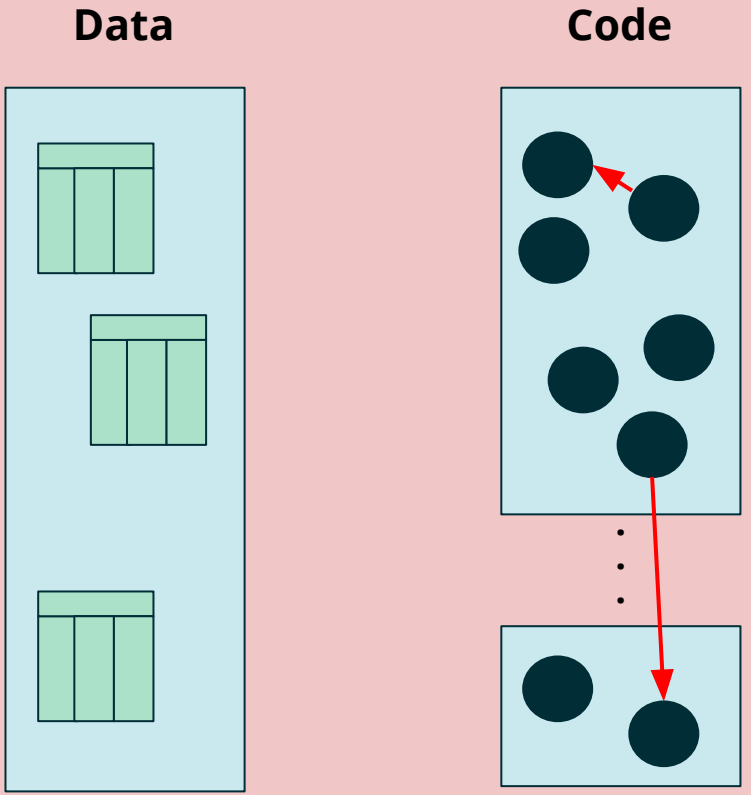
APP DATABASE

APP DATABASE

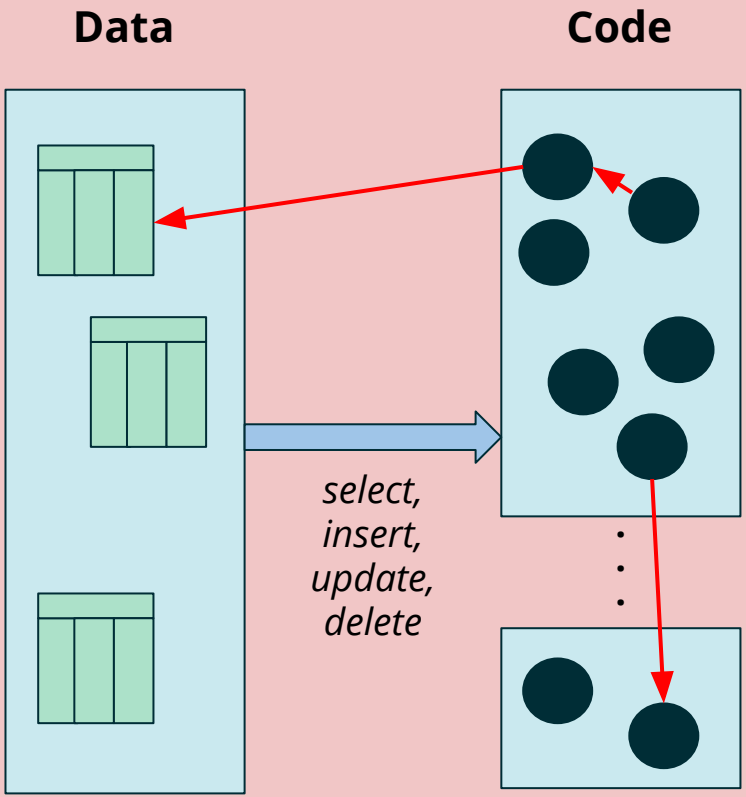
Data



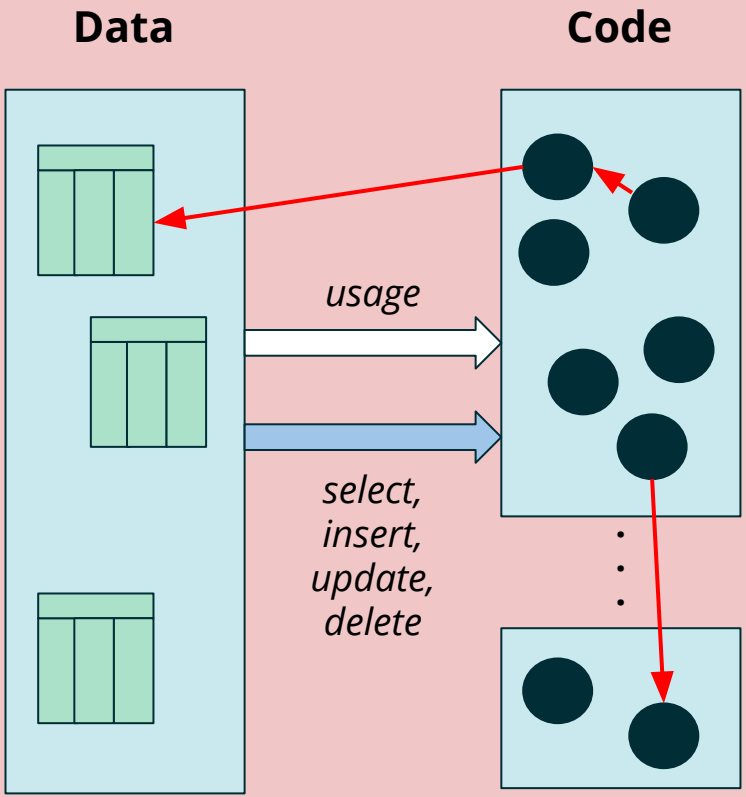
APP DATABASE



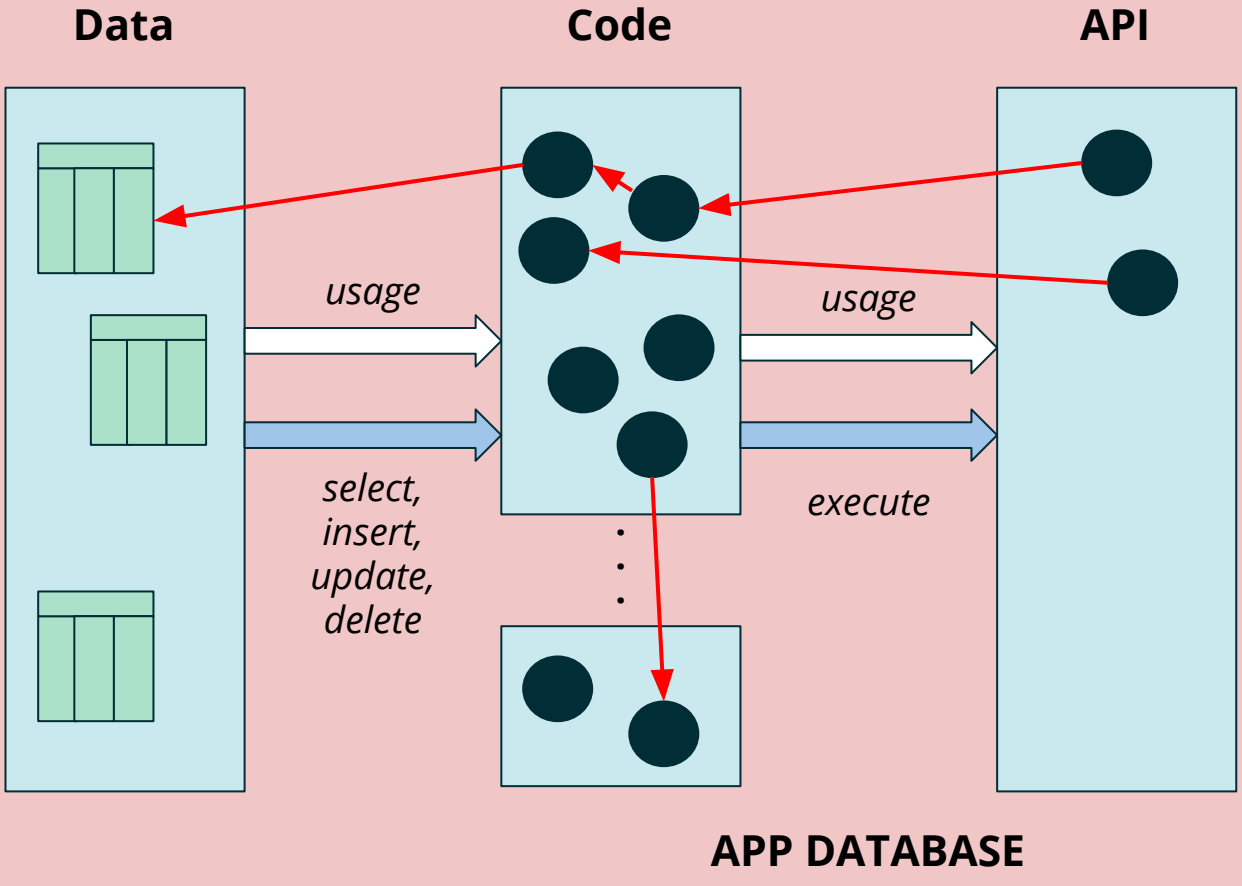
APP DATABASE

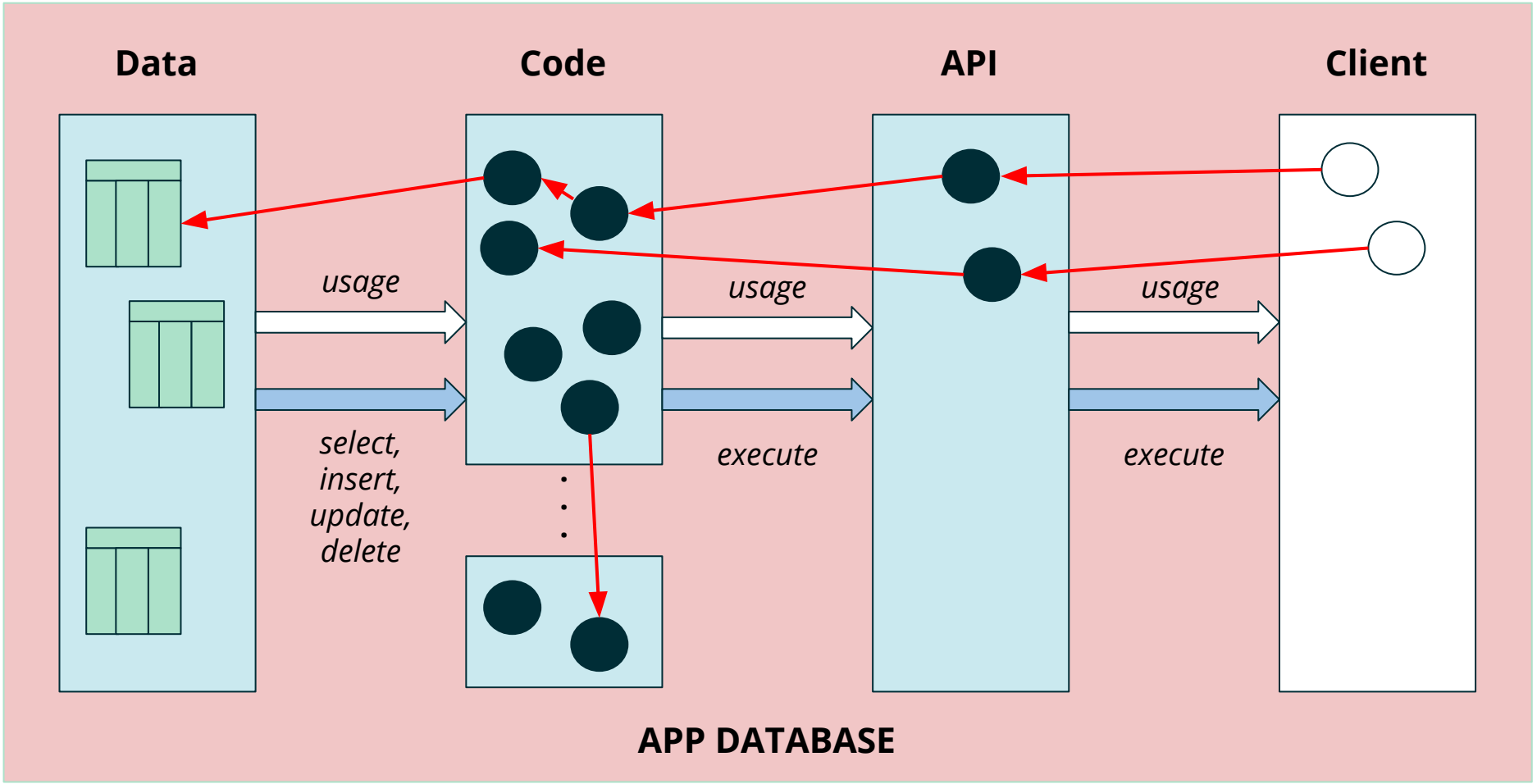


APP DATABASE

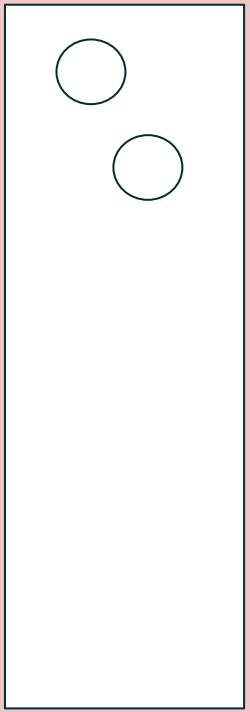


APP DATABASE



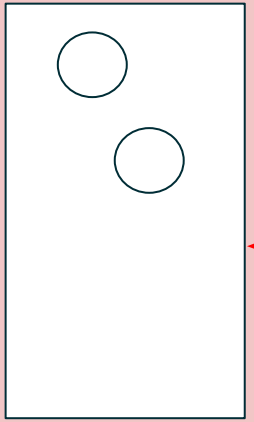


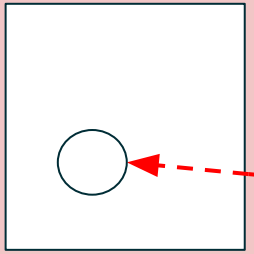
Client



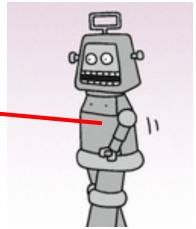
APP DATABASE

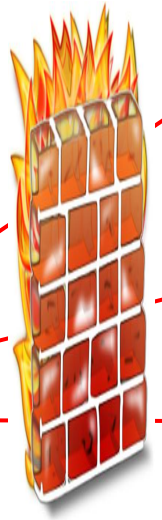
Client





APP DATABASE





Again: Use stored procedures to encapsulate the RDBMS's functionality behind an impenetrable hard shell

Setting up the purpose-agnostic foundation and superstructure

The superuser lays the foundation

The superuser's foundation work

```
-- Deliberate least privilege.
-- Users exist in cluster scope, not database scope.
-- So don't want app_ower to be able to create users.

\c yugabyte yugabyte

create user app_owner login password 'app';

create database app owner app_owner;

grant create on database app to app_owner;

\c app yugabyte

drop schema public;
```

The superuser's foundation work—cont.

```
-- Now create the users that will own schema-objects.

\c app yugabyte

create user data login password 'data';
grant create on database app to data;

create user code login password 'code';
grant create on database app to code;

create user api login password 'api';
grant create on database app to api;

create user client login password 'client';
```

The application administrators erect the superstructure

The application administrators take over

```
-- Create and populate data schema.

\c app data

create schema data authorization data;
grant usage on schema data to code;
alter user data set search_path = "data";

\i create_data_objects.sql
```

The application administrators take over—cont.

```
-- Create and populate the code schema.

\c app code

create schema code authorization code;
grant usage on schema code to api;
alter user code set search_path = "code";

\i create_code_objects.sql
```

The application administrators take over—cont.

```
-- Create and populate the api schema.

\c app api

create schema api authorization api;
grant usage on schema api to client;
alter user code set search_path = "api";

\i create_api_objects.sql
```

The application administrators take over—cont.

```
-- Create the schemaless client.

\c app client

alter user code set search_path = "api";
```

An example application:

Maintain masters and their details

The data objects

create_data_objects.sql

```
create table data.masters(
  mk int
  generated always as identity
  constraint masters_pk primary key
  constraint masters_mk_chk check(mk > 0),

  v varchar(30) not null
  constraint masters_v_unq unique);


revoke all on data.masters from public;

grant select, insert, update, delete on data.masters to code;
```

create_data_objects.sql — cont.

```
create table data.details(
  mk int,

  dk int
  generated always as identity
  constraint details_dk_chk check(dk > 0),

  v varchar(30) not null,
  constraint details_pk primary key(mk, dk),

  constraint details_fk foreign key(mk)
    references data.masters(mk)
    match full
    on delete cascade
    on update restrict,

  constraint details_mk_v_unq unique(mk, v));

revoke all on data.details from public;
grant select, insert, update, delete on data.details to code;
```

The code objects

create_code_objects.sql

```
create procedure code.insert_master_and_details(
  m in varchar,
  ds in varchar[])
  language plpgsql
  security definer
as $$
declare
  ...
begin
  ...
end;
$$;
```

create_code_objects.sql — cont.

```
begin
  -- Disallow master with no details.
  if cardinality(ds) < 1 then
    raise exception 'Cannot insert master "%" with no details.', m
    using errcode = '99998';
  end if;

  insert into data.masters(v)
     values(m) returning mk into new_mk;

  foreach d in array ds loop
    begin
       insert into data.details(mk, v) values(new_mk, d);
    exception when unique_violation then
      raise exception 'Master "%" already has detail "%"', m, d
      using errcode = '99998';
    end;
  end loop;
end;
```

create_code_objects.sql – cont.

```
revoke all on procedure code.insert_master_and_details(
  m in varchar,
  ds in varchar[])
from public;



grant execute on procedure code.insert_master_and_details(
  m in varchar,
  ds in varchar[])
to api;
```

create_code_objects.sql – cont.

```
create table code.result_template(m varchar(10), ds varchar[]);
revoke all on code.result_template from public;

create function code.report_master_and_details(the_mv in varchar)
  returns record
  language plpgsql
  security definer
as $$
declare
  ...
  result code.result_template%rowtype;
  ...
begin
  ...
end;
$$;
```

create_code_objects.sql — cont.

```
declare
  row record;
  result code.result_template%rowtype;
  ds varchar[] := array[]::varchar[];
  n int not null := 0;
  first_m varchar(10) not null := '?';
begin
  for row in (
    -- left outer join to allow for the possibility that master has no details.
    select m.v as mv, d.v as dv
    from data.masters m left outer join data.details d using (mk)
    where m.v = the_mv
    order by 1, 2)
  loop
    n := n + 1;
    if n = 1 then
      first_m := row.mv;
    end if;
    ds := ds||row.dv;
  end loop;


  if n < 1 then
    raise exception 'The key "%" does not exist', the_mv
    using errcode = '99998';
  end if;


  result.m := the_mv;
  result.ds := ds;
  return result;
end;
```

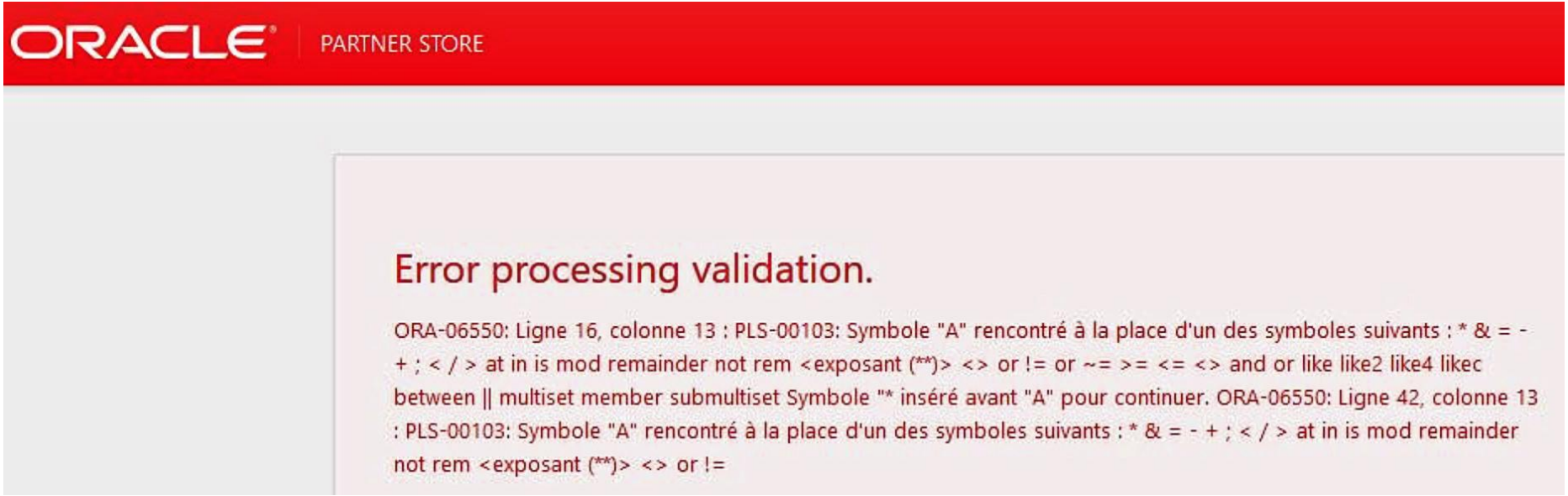
create_code_objects.sql – cont.

```
revoke all on function code.report_master_and_details(
  the_mv in varchar)
from public;



grant execute on function code.report_master_and_details(
  the_mv in varchar)
to api;



-- Not yet supported in YugabyteDB Version 2.0.1
alter function code.report_master_and_details(
  the_mv in varchar)
set search_path = code;
```

The api objects

create_api_objects.sql

```
create procedure api.insert_master_and_details(
  m in varchar,
  ds in varchar[])
  language plpgsql
  security definer
as $$
declare
  ...
begin
  ...
end;
$$;
```



create_api_objects.sql — cont.

```
declare
  the_sqlstate      varchar not null := '?';
  the_error_message varchar not null := '?';
begin
  call code.insert_master_and_details(m, ds);
exception when others then
  -- 15 lines of bolerplate text. But there's no better way.
 if sqlstate = '99998' then
    -- Expected but regrettable error
    raise exception 'Expected error: %', sqlerrm
    using errcode = '99998';
  else
    -- Unexpected error
    get stacked diagnostics
      the_sqlstate      = returned_sqlstate,
      the_error_message = message_text;

    raise exception '%',
      api_utils.formatted_error_info(the_sqlstate, the_error_message)
      using errcode = '99999';
  end if;
end;
```

Exercising the client functionality

```
-- Demo time
```

```
\c app client
```

```
select schema_name from information_schema.schemata;
```

```
    schema_name
--------------------
 information_schema
 pg_catalog
 api
```

```
\df
```

```
 Schema |                Name                |      Result data type      |            Argument data types             | Type
--------+------------------------------------+----------------------------+--------------------------------------------+------
 api    | insert_master_and_details          |                            | m character varying, ds character varying[] | proc
 api    | report_master_and_details          | record                     | the_mv character varying                   | fun
```

```
-- Demo time

call insert_master_and_details(
  'Mary',
  array['soap', 'towel', 'toothbrush', 'shampoo']);



select report_master_and_details('Mary')
  as "Mary and her details";
```

```
          Mary and her details
----------------------------------------
 (Mary,"{shampoo,soap,toothbrush,towel}")
```

```
-- Demo time

call insert_master_and_details(
  'Joan',
  array[]::varchar[]);

ERROR:
Expected error: Cannot insert master "Joan" with no details.

call insert_master_and_details(
  'Mary',
  array['screwdriver', 'saw']);

ERROR:
Something appears to have gone wrong. Contact Support with this info:
  Incident timestamp: 23:09:03 23-Oct-2019
  sqlstate:           23505
  error message:      duplicate key value violates
                      unique constraint "masters_v_unq"
```

```
-- Demo time

\c app data


select mk, m.v, d.v, d.dk
from data.masters m
     left outer join
     data.details d using (mk)
order by mk, d.dk;


delete from data.masters
where v in ('Mary', 'John', 'Alice', 'Bill', 'Joan');
```

```
-- Demo time

\c app client

show server_version;

create table api.t(n int);
create table pg_catalog.t(n int);
create table information_schema.t(n int);

create procedure p()
  language plpgsql
  security definer
as $$
begin
  raise info 'hello from api.p()';
end;
$$;
```

Summary

Benefits of encapsulating the database behind a hard shell API

- **Correctness**
  - Separation of skills and concerns
  - The right experts own their own tersely coupled modules
  - Type safety
- **Security**
  - Limit visibility of objects (directly and via error messages)
  - Implicit SQL-injection proofing
- **Performance**
  - Minimization of client-server round trips
  - Prepare-execute paradigm with no coding effort

Questions?

Download
download.yugabyte.com

Join Slack Discussions
yugabyte.com/slack

Star on GitHub
github.com/YugaByte/yugabyte-db