HPL/SQL Stored Procedures with an HMS storage backend

Introduction

The goal of this document is to investigate a potential HMS storage backend behind HPL/SQL stored procedures. Currently HPL/SQL stores everything in text files and users can include code from one file into the other using the INCLUDE <path> statement.

This makes it difficult to manage and keep track of the stored procedures. Storing in a centralized place (like in Hive MetaStore) would make these things easier.

Metadata

Storing the metadata (parameters, return value and types) about functions (in a structured way) as well as the code would be useful for doing semantic analysis, validating function arity or doing basic type checking.

However if we want to support multiple programming languages in the future and not just PL/SQL we should also consider different calling semantics.

Most programming languages support functions with positional arguments and with an optional single return value.

```
<return_type> func_name(type arg1, type arg2, ... type argN);
```

This is the most common one among mainstream programming languages but there are some others:

1. **Positional parameters (most common)**
2. **Output parameters (HPL/SQL already has this so this needs to be supported).**
3. Named parameters (less common, Python, Ruby 2.0, Smalltalk, ObjcC)
4. Varargs (newer version of JavaScript has this)
5. Default values (JavaScript has this since ES6)
6. Multiple return value (PL/SQL only knows this via custom data type, Python, Go, FORTH however support this natively)

- If we only want to cover PL/SQL then we need to have a fixed number of positional arguments with output parameters and a single return type **(1. 2.)**.

- If we want to cover recent JavaScript versions as well, we'll need a variable number of positional arguments with optional default values and a single return type **(1. 4. 5)**.

Tables

Metadata and the source code of a stored procedure can be stored in HMS's database.

Stored procedure metadata and source code:

```
CREATE TABLE "STORED_PROCS" (
    "SP_ID" bigint NOT NULL,       -- primary key
    "CREATE_TIME" bigint NOT NULL,
    "LAST_ACCESS_TIME" bigint NOT NULL,
    "DB_ID" bigint,                -- foreign key -> DB (default is default)
    "NAME" VARCHAR(256) NOT NULL,  -- Function Name
    "OWNER" character varying(767) DEFAULT NULL::character varying,
    "SOURCE" text,                 -- uncompiled source code/serialized AST
    "ARITY" INTEGER NOT NULL,      -- Number of arguments
    "LANG" VARCHAR(128) NOT NULL,  -- Implementation language (HPLSQL/JS)
    "RETURN_TYPE" VARCHAR(128),    -- If any
    PRIMARY KEY ("SP_ID")
);
```

Stored procedure arguments for positional parameters (with optional default values, out parameters and varargs):

There is a one-to-many relationship between `STORED_PROCS` and `SP_POS_ARGS`.

```
CREATE TABLE "SP_POS_ARGS" (
  "POS" integer NOT NULL,                 -- position 0..255, primary key
  "SP_ID" bigint NOT NULL,                -- foreign key -> STORED_PROC
  "TYPE" VARCHAR(128) NOT NULL,           -- data type
  "NAME" VARCHAR(256) NOT NULL,           -- parameter name
  "OUT" BOOLEAN NOT NULL DEFAULT false,   -- if it's an output parameter
  "DEFAULT_VALUE" VARCHAR(256),           -- default value
  "VARARG" BOOLEAN NOT NULL DEFAULT false, -- true if can be repeated n times
  PRIMARY KEY ("POS")
);
```

Note that Postgres stores the arguments about its stored procedures in the same table (instead of 2 different tables) in array columns: https://www.postgresql.org/docs/9.3/catalog-pg-proc.html

Maybe we can do it similarly but since HMS can be used with multiple database backends it might not be an option.

HMS API

A few new HMS API endpoints need to be added:

- `void createStoredProc(String dbName, CreateStoredProcRequest req);`
- `void deleteStoredProc(String dbName, String procedureName);`
- `List<String> listStoredProcs(String dbName);`
- `ProcMeta getStoredProc(String dbName, String procedureName);`

A stored procedure is binded to a database, not sure if we want to add another level for namespacing or if this is sufficient.

The `STORED_PROCS` and SP_POS_ARGS tables can be populated using the content of CreateStoredProcRequest.

HPL/SQL modifications

Defining a procedure

Running the following would add a new row to STORED_PROCS table and 2 new rows in SP_POS_ARGS table, via HMS.`createStoredProc(..);`

```
CREATE FUNCTION hello1(p1 STRING, OUT outp2 STRING)
RETURNS STRING
BEGIN
 ..
END;
```

Would result:

```
INSERT INTO "STORED_PROCS" VALUES(
    1,      -- primary key
    <NOW>,
    <DB_ID>,
    <NOW>,
    <CURRENT_USER>,
    ‘CREATE FUNCTION hello1(IN name STRING, OUT result STRING) BEGIN .. END’
```

```
    2,      -- Number of arguments
    "HPL/SQL",
    STRING
);
```

Instead of storing the full definition as a string it might be enough to only store the body since metadatas are already stored in a structured way. And it might be better to store a serialized version of an AST instead of the text.

And for p1:

```
INSERT INTO "SP_POS_ARGS" (
  0,                                -- position 0..255
  1,                                -- foreign key -> STORED_PROC
  "STRING,                          -- data type
  "p1",                             -- parameter name
  false,                            -- if it’s an output parameter
  null,                             -- default value
  false,                            -- vararg, can be repeated n times
);
```

And for outp2:

```
INSERT INTO "SP_POS_ARGS" (
  1,                                -- position 0..255
  1,                                -- foreign key -> STORED_PROC
  "STRING,                          -- data type
  "outp2",                          -- parameter name
  true,                             -- if it’s an output parameter
  null,                             -- default value
  false,                            -- vararg, can be repeated n times
);
```

Redefining an existing procedure should drop the old definition and recreate it completely.

Calling a procedure

Calling the procedure `hello1` would trigger the following sequence:

1. `ProcMeta procMeta = HMS.getStoredProc(<CURRENTLY_SELECTED_DB>, ‘hello1’);`
2. `validate(actualArguments, formalParameters(procMeta));`

3. eval(procMeta.source)

The procedure should be registered under the AST into the running interpreter process to prevent further interactions to HMS on repeated calls.

Right now this can be integrated into HPL/SQL but in the future we might want to move it to HiveServer2.

Packages

HPL/SQL supports packages which are second class in the language (cannot be stored in variables or passed around, single implementation, no dynamic dispatch), however the implementation of a package can be changed globally.

```
create or replace package users as
  session_count int := 0;
  function get_count() return int;
  procedure add(name varchar(100));
end;
```

Package body:

```
create or replace package body users as
  function get_count() return int
  is
  begin
    return session_count;
  end;
  procedure add(name varchar(100))
  is
  begin
    -- ...
    session_count = session_count + 1;
  end;
end;
```

This would require adding a PACKAGE table and a (nullable) FK in the STORED_PROCS table.

```
CREATE TABLE "STORED_PROCS" (
    "SP_ID"  bigint NOT NULL,      -- primary key
    "PKG_ID” bigint                -- foreign key -> packages
```

```
    ...
);

CREATE TABLE "PKG_STORED_RPOCS" (
    "PKG_ID"  bigint NOT NULL,      -- primary key
    "NAME"    VARCHAR(256),
    ...
);
```