# Stratum

Apr 19, 2020

# Contents

Stratum is a general idea and concept for loading and invoking stored procedures (in a database) from an application using automatically generated wrappers.

Currently, the following Stratum projects are available (follow the link to navigate to the project specific documentation):

| Language | MySQL & MariaDB | PostgreSQL | SQLite | SQL Server |
|---|---|---|---|---|
| PHP | PhpStratum MySQL | | PhpStratum SQLite PDO | |
| Python | PyStratum MySQL | PyStratum pgSQL | | PyStratum MSSQL |

In this documentation we explain the general idea and concept that all above Stratum projects have in common.

CHAPTER 1

# Introduction

Many database systems have the notion of stored procedures, stored functions, stored routines, and/or stored packages. Some database system vendors distinguish stored procedures and stored functions, some database systems have stored functions only, whilst other database systems have stored packages, and some database systems use the term stored routines to amalgamate both stored procedures and stored functions.

Throughout this documentation we use the term stored procedures in the broadest possible meaning: the concept of a piece (user defined) code in a database that can be invoked from an application (regardless whether this piece of code is actual is procedure, function, or part of a package).

This documentation is about Stratum. We will not discuss the advantages and drawbacks of using stored procedures. You are here, so you are on the right track.

CHAPTER 2

# Concepts

The concepts of a Stratum project comprises:

- the functionalities of the Stratum project,
- the so called the designation type of stored procedures,
- and constants based and column width and labels.

## 2.1 Functionalities of a Stratum Project

A Stratum project must provide the following functionalities:

- Loading: Maintain the routines in the target database:
  - When a new source file with a stored procedure is available the stored procedure must be loaded in to the target database.
  - When a source file has been modified or some other relevant setting, configuration variable, etc. has been modified the stored procedure must be reloaded in to the database.
  - When a source file of a stored procedure is no longer available the stored procedure must be removed from the database.
- Wrapping: Generate code for invoking the stored procedures. The generated code must relieve the developers of the application of the hustle of invoking a stored procedure such as (depending on the programing language and database system):
  - Parameter binding.
  - Preventing SQL injection.
  - Character set translating.
  - Checking for error codes or statuses and error and exception handling.
  - Validating the result set of a store routine against is designation type, see section *Designation Type of Stored Procedures*.

- Constants: Generate code with constants (or some other mechanism) based on widths of table columns and labels, see section *Constants*.

## 2.2 Designation Type of Stored Procedures

Depending and the database system, programing language and actual type of the stored procedure (e.g. stored function) stored procedures return result sets or even multiple result sets. However,

- A stored procedure that counts the current number of employees in a department returns a result set with one row with one column only.
- A stored procedure that selects the details (e.g. name, given name, employee number, and department) of an employee given a technical ID returns a result set with one row only. Moreover, if we impose that this stored procedure is invoked with valid technical IDs only the result set is expected to have one and only one row. If the stored procedure returns a result set with zero rows the application has a bug (e.g. passed a wrong technical ID). If the stored procedure returns a result set with two or more rows the stored procedure has a bug (e.g. a forgotten (join) condition).
- A stored procedure that selects the details of all employees of a department returns a result set with multiple rows.

As an application developer, you don't want for each stored procedure like the first example take the first row from (depending on the programming language) an array, a list, a set, or iterate only once over a cursor, then take the first column and validate that the found value is indeed an integer.

As an application developer, you don't want for each stored procedure like the second example validate that the result set has exactly one row and throw an exception if the number of rows is not one, then take the first row from (depending on the programming language) an array, a list, a set, or iterate only once over a cursor.

This observation has led to the so called designation type of a stored procedure. The designation type of a stored procedure does not affect the stored procedure but only the (automatically generated) wrapper for invoking the stored procedure. Since the concept of designation types is unknown to database systems the designation type of a stored procedure must be documented in a comment in the source of the stored procedure.

We distinguish the following (basic) designation types:

- **function** A stored *procedure* with designation type function is in fact a stored *function* and returns a single value only.
- **row0** A stored procedure with designation type row0 selects zero or one row only. When the stored procedure selects two or more rows an exception must be thrown.
- **row1** A stored procedure with designation type row1 selects one and only one row only. When the stored procedure selects zero, two or more rows an exception will be thrown.
- **rows** A stored procedure with designation type rows selects zero, one, or more rows. The total result set must fit into the memory.
- **singleton0** A stored procedure with designation type singleton0 selects zero or one row only and the selected row has one column only. In the application the selected value must be returned as single value (not an array, dictionary). When the stored procedure selects two or more rows an exception will be thrown.
- **singleton1** A stored procedure with designation type singleton1 selects one and only one row and the selected row has one column only. In the application the selected value must be returned as single value (not an array, dictionary). When the stored procedure selects zero, two or more rows an exception will be thrown.
- **void** A stored procedure with designation type void does not select any rows.

Other designation types are:

- **bulk** A stored procedure with designation type bulk is intended for stored procedures that select a large number of rows that might not fit into the memory.
- **bulk_insert** A stored procedure with designation type bulk_insert is intended for stored procedures that insert multiple rows into a table.
- **hidden** A stored procedure with designation type hidden is a stored procedure that is called by other stored procedures and is not intended to be invoked from the application.
- **log** A stored procedure with designation type log returns multiple result sets with typically one row with one column. These values are logged to a logger or (with timestamp) on the standard output.
- **map** A stored procedure with designation type map select zero, one, or more rows but the selected rows have two columns only. The wrapper of the stored procedure will not return the result set but map (using an appropriate data structure in the programing language) from the first column to the second column.
- **table** A stored procedure with designation type rows select zero, one, or more rows but the wrapper of the stored procedure does not return those rows but prints then in a table format to the standard output.

A Stratum project might not implement all above designation types or implement other designation types as well.

## 2.3 Constants

In a Stratum project we distinguish constants based on length of column types and constants based on (integer) primary keys.

Although, we speak of constants in a Stratum project they can be implemented with other means that constants in the programming language. An optimal implementation in the programing language is preferred[1].

### 2.3.1 Column Type

Most database systems have column types with length constraint. For example:

- `varchar(40)`
- `int(4)`

Suppose in your application (either a website, a GUI, or some other kind of program) the length of street names is limited to 40 characters, than:

- The optimal column type for street name is `varchar(40)`[2].
- An input field for a street name should have length 40 too (or limit user's input to 40 characters).
- Your application has to validate user or external input of street names has 40 or less characters.

One could define in your application a constant for each (input) field stored in a column with a length constraint and (hope) that the constants defined in your application and the actual column types in your database are aligned. A much better solution is to automatically define constants in your application based on the actual length of columns. Moreover, when the maximum length of a column in the database is altered the corresponding constant is altered automatically too.

[1] Moreover, in a compiled language one cannot use constants because the values depend on the actual values of the data and metadata of the database.

[2] One might argue that a column type `varchar(255)` or even `varchar(2048)` will suffice too or even is better because most database systems only require the actual length of the street name when the table is stored on the filesystem and when in your application the maximum length of street names is increased to, say, 80 characters no database change is required. However, most database systems when reading (a row) will reserve the maximum possible length of column in the memory. Hence, a street name of 40 characters stored in a `varchar(2048)` column with a 32 bits UTF-8 encoding will require 8192 bytes in the memory where only 160 bytes are required. This is an overhead of more than 50 times. This will impact the performance of your database in terms of cache hits and concurrency. Conclusion, you must align your maximum lengths in your application and the column types in your database.

### 2.3.2 Primary Key

Suppose, the database of your application has a reference table and depending on the actual values in the rows of this table business logic has to be implemented. You have two possible solutions:

- Use a meaningful secondary key that will not changes.
- Use a constant based on the meaningless primary key.

For example, consider the following reference table:

| cnt_id | cnt_iso_abbr | cnt_name | cnt_label |
|---|---|---|---|
| 1 | – | `null` | `C_CNT_ID_NONE` |
| 2 | NL | Netherlands | `C_CNT_ID_NL` |
| 3 | BE | Belgium | `null` |
| 4 | DE | Germany | `null` |
| 5 | US | United States of America | `null` |

One could use the ISO Alpha-2 country code (stored in column `cnt_iso_abbr`) for implementing business logic rules involving countries. Alternatively, one could generated automatically constants based on the last column `cnt_label` (the name of the constant is taken from `cnt_label` and its value from `cnt_id`) and use these constants for implementing business logic rules involving countries.

This last method will also work with reference tables without a meaningful secondary key or where the meaningful secondary key is likely to change.

A Stratum project will allow you define constants based on the actual values of primary (integer) keys and another column with the name of the constant.

CHAPTER 3

# Benefits

In this chapter we discuss the advantages of using a Stratum project (we will not discuss the advantages of using store procedures) over manually coding software for invoking stored procedures from your application.

Advantages of using a Stratum project (in no particular order) are:

- No boiler templating. Using a Stratum project relieves you from boiler templating your stored procedure code from dropping the stored procedure if it already exists or setting delimiters. Dropping or replacing the stored procedure or setting delimiters is done automatically by Stratum.
- Stratum generates wrapper methods automatically based on the metadata retrieved from the database. Hence, the wrapper will have the right number and the right types of arguments corresponding with the arguments of the stored procedure. Also, the return type of the wrapper is automatically determined based in the designation type of the stored procedure. Hence, when a stored procedure has been modified:
  - the number of arguments has changed,
  - the type of an argument has changed,
  - the designation type has changed.

  The wrapper for the stored procedure will be automatically changed to align with the stored procedure. When in your code the stored procedure is invoked with wrong number of arguments, an argument of the wrong type, or expecting a different return type, automated code inspection, IDE, or compiler will notify you about this mismatch.

  When you code manually software for invoking stored procedure, you might oversee the changed arguments or return type and will be unaware about this issue till someone runs your code.
- Saving time and cost. Wrappers for invoking stored procedures from your application are generated automatically saving you coding manually. Also, when you modify or add a stored procedure, the stored procedure will be loading into your database automatically.
- Automatically loading of stored procedures. Stored procedures are loaded into the database automatically by Stratum. Hence, you don't have to write scripts for loading stored procedures into your database. Both for your development database or the production database. Also, the correct version of the stored procedures are always loaded into the database, no stored procedure is overseen.

- Constants based on data and metadata stored in database. In your application you can use constants based on the metadata or data of your database.
- Consistent mapping between types in the programming language and database types.
- Invoking a stored procedure is just as simple as calling any other method of a class.
- No SQL code in your application code base.
- Improved security. Your application requires execution rights on stored procedures only. Tables, stored procedures and other database entities can be (highly recommended) owned by another account than the account your application is using for accessing the database.
- Automatically generated wrappers for invoking stored procedures will protect your application against SQL injection attacks.