NAME

SPI_prepare - prepare a statement, without executing it yet

SYNOPSIS

SPIPlanPtr SPI_prepare(const char * *command*, int *nargs*, Oid * *argtypes*)

DESCRIPTION

SPI_prepare creates and returns a prepared statement for the specified command, but doesn't execute the command. The prepared statement can later be executed repeatedly using **SPI_execute_plan**.

When the same or a similar command is to be executed repeatedly, it is generally advantageous to perform parse analysis only once, and might furthermore be advantageous to re-use an execution plan for the command. **SPI_prepare** converts a command string into a prepared statement that encapsulates the results of parse analysis. The prepared statement also provides a place for caching an execution plan if it is found that generating a custom plan for each execution is not helpful.

A prepared command can be generalized by writing parameters ($1, $2, etc.) in place of what would be constants in a normal command. The actual values of the parameters are then specified when **SPI_execute_plan** is called. This allows the prepared command to be used over a wider range of situations than would be possible without parameters.

The statement returned by **SPI_prepare** can be used only in the current invocation of the C function, since **SPI_finish** frees memory allocated for such a statement. But the statement can be saved for longer using the functions **SPI_keepplan** or **SPI_saveplan**.

ARGUMENTS

const char * *command*
: command string

int *nargs*
: number of input parameters ($1, $2, etc.)

Oid * *argtypes*
: pointer to an array containing the OIDs of the data types of the parameters

RETURN VALUE

SPI_prepare returns a non-null pointer to an SPIPlan, which is an opaque struct representing a prepared statement. On error, NULL will be returned, and *SPI_result* will be set to one of the same error codes used by **SPI_execute**, except that it is set to SPI_ERROR_ARGUMENT if *command* is NULL, or if *nargs* is less than 0, or if *nargs* is greater than 0 and *argtypes* is NULL.

NOTES

If no parameters are defined, a generic plan will be created at the first use of **SPI_execute_plan**, and used for all subsequent executions as well. If there are parameters, the first few uses of **SPI_execute_plan** will generate custom plans that are specific to the supplied parameter values. After enough uses of the same prepared statement, **SPI_execute_plan** will build a generic plan, and if that is not too much more expensive than the custom plans, it will start using the generic plan instead of re-planning each time. If this default behavior is unsuitable, you can alter it by passing the CURSOR_OPT_GENERIC_PLAN or CURSOR_OPT_CUSTOM_PLAN flag to **SPI_prepare_cursor**, to force use of generic or custom plans respectively.

Although the main point of a prepared statement is to avoid repeated parse analysis and planning of the statement, PostgreSQL will force re-analysis and re-planning of the statement before using it whenever database objects used in the statement have undergone definitional (DDL) changes since the previous use of the prepared statement. Also, if the value of search_path changes from one use to the next, the statement will be re-parsed using the new *search_path*. (This latter behavior is new as of PostgreSQL 9.3.) See **PREPARE**(7) for more information about the behavior of prepared statements.

This function should only be called from a connected C function.

SPIPlanPtr is declared as a pointer to an opaque struct type in spi.h. It is unwise to try to access its contents directly, as that makes your code much more likely to break in future revisions of PostgreSQL.

The name SPIPlanPtr is somewhat historical, since the data structure no longer necessarily contains an execution plan.