## NAME

SPI_cursor_parse_open - set up a cursor using a query string and parameters

## SYNOPSIS

Portal SPI_cursor_parse_open(const char *_name_,
const char *_command_,
const SPIParseOpenOptions * _options_)

## DESCRIPTION

**SPI_cursor_parse_open** sets up a cursor (internally, a portal) that will execute the specified query string. This is comparable to **SPI_prepare_cursor** followed by **SPI_cursor_open_with_paramlist**, except that parameter references within the query string are handled entirely by supplying a ParamListInfo object.

For one-time query execution, this function should be preferred over **SPI_prepare_cursor** followed by **SPI_cursor_open_with_paramlist**. If the same command is to be executed with many different parameters, either method might be faster, depending on the cost of re-planning versus the benefit of custom plans.

The *options*->*params* object should normally mark each parameter with the PARAM_FLAG_CONST flag, since a one-shot plan is always used for the query.

The passed-in parameter data will be copied into the cursor's portal, so it can be freed while the cursor still exists.

## ARGUMENTS

const char * *name*
name for portal, or NULL to let the system select a name

const char * *command*
command string

const SPIParseOpenOptions * *options*
struct containing optional arguments

Callers should always zero out the entire *options* struct, then fill whichever fields they want to set. This ensures forward compatibility of code, since any fields that are added to the struct in future will be defined to behave backwards-compatibly if they are zero. The currently available *options* fields are:

ParamListInfo *params*

data structure containing query parameter types and values; NULL if none

int *cursorOptions*
integer bit mask of cursor options; zero produces default behavior

bool *read_only*
true for read-only execution

**RETURN VALUE**

Pointer to portal containing the cursor. Note there is no error return convention; any error will be reported via **elog**.