**pgTAP 1.0.0**

pgTAP es un marco de prueba de unidad para PostgreSQL escrito en PL / pgSQL y PL / SQL. Incluye una colección completa de funciones de aserción de emisión de [TAP](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://testanything.org/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhg8U-xa2q_hsFIrYXn-J5QtJKCbew) , así como la capacidad de integración con otros marcos de prueba de emisión de TAP. También se puede utilizar en el estilo de prueba xUnit.

**Contenido**

* [Sinopsis](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#synopsis)
* [Instalación](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#installation)
  + [Probando pgTAP con pgTAP](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#testingpgtapwithpgtap)
  + [Añadiendo pgTAP a una base de datos](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#addingpgtaptoadatabase)
* [PgTAP Test Scripts](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#pgtaptestscripts)
  + [Usando pg\_prove](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#usingpg_prove)
* [Usando pgTAP](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#usingpgtap)
  + [Me encanta cuando un plan viene junto](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#iloveitwhenaplancomestogether)
  + [¡Qué unidad tan dulce!](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#whatasweetunit)
  + [Descripciones de Pruebas](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#testdescriptions)
    - [xunidad nombres de funciones](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#xunitfunctionnames)
  + [Estoy bien, no estás bien](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#imokyourenotok)
    - [ok()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#ok)
    - [is()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is)
    - [isnt()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isnt)
    - [matches()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#matches)
    - [imatches()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#imatches)
    - [doesnt\_match()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#doesnt_match)
    - [doesnt\_imatch()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#doesnt_imatch)
    - [alike()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#alike)
    - [ialike()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#ialike)
    - [unalike()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#unalike)
    - [unialike()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#unialike)
    - [cmp\_ok()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#cmp_ok)
    - [pass()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#pass)
    - [fail()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#fail)
    - [isa\_ok()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isa_ok)
* [Persiguiendo tu consulta](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#pursuingyourquery)
  + [El error es humano](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#toerrorishuman)
    - [throws\_ok()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#throws_ok)
    - [throws\_like()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#throws_like)
    - [throws\_ilike()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#throws_ilike)
    - [throws\_matching()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#throws_matching)
    - [throws\_imatching()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#throws_imatching)
    - [lives\_ok()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#lives_ok)
    - [performs\_ok()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#performs_ok)
    - [performs\_within()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#performs_within)
  + [¿Te identificas?](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#canyourelate)
    - [results\_eq()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#results_eq)
    - [results\_ne()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#results_ne)
    - [set\_eq()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#set_eq)
    - [set\_ne()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#set_ne)
    - [set\_has()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#set_has)
    - [set\_hasnt()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#set_hasnt)
    - [bag\_eq()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#bag_eq)
    - [bag\_ne()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#bag_ne)
    - [bag\_has()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#bag_has)
    - [bag\_hasnt()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#bag_hasnt)
    - [is\_empty()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_empty)
    - [isnt\_empty()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isnt_empty)
    - [row\_eq()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#row_eq)
* [Las cosas del esquema](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#theschemathings)
  + [Me opongo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#iobject)
    - [tablespaces\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#tablespaces_are)
    - [schemas\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#schemas_are)
    - [tables\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#tables_are)
    - [partitions\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#partitions_are)
    - [foreign\_tables\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#foreign_tables_are)
    - [views\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#views_are)
    - [materialized\_views\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#materialized_views_are)
    - [sequences\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#sequences_are)
    - [columns\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#columns_are)
    - [indexes\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#indexes_are)
    - [triggers\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#triggers_are)
    - [functions\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#functions_are)
    - [roles\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#roles_are)
    - [users\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#users_are)
    - [groups\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#groups_are)
    - [languages\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#languages_are)
    - [opclasses\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#opclasses_are)
    - [rules\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#rules_are)
    - [types\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#types_are)
    - [domains\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#domains_are)
    - [enums\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#enums_are)
    - [casts\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#casts_are)
    - [operators\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#operators_are)
    - [extensions\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#extensions_are)
  + [Tener o no tener](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#tohaveorhavenot)
    - [has\_tablespace()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_tablespace)
    - [hasnt\_tablespace()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_tablespace)
    - [has\_schema()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_schema)
    - [hasnt\_schema()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_schema)
    - [has\_relation()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_relation)
    - [hasnt\_relation()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_relation)
    - [has\_table()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_table)
    - [hasnt\_table()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_table)
    - [has\_view()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_view)
    - [hasnt\_view()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_view)
    - [has\_materialized\_view()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_materialized_view)
    - [hasnt\_materialized\_view()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_materialized_view)
    - [has\_inherited\_tables()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_inherited_tables)
    - [hasnt\_inherited\_tables()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_inherited_tables)
    - [is\_ancestor\_of()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_ancestor_of)
    - [isnt\_ancestor\_of()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isnt_ancestor_of)
    - [is\_descendent\_of()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_descendent_of)
    - [isnt\_descendent\_of()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isnt_descendent_of)
    - [has\_sequence()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_sequence)
    - [hasnt\_sequence()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_sequence)
    - [has\_foreign\_table()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_foreign_table)
    - [hasnt\_foreign\_table()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_foreign_table)
    - [has\_type()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_type)
    - [hasnt\_type()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_type)
    - [has\_composite()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_composite)
    - [hasnt\_composite()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_composite)
    - [has\_domain()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_domain)
    - [hasnt\_domain()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_domain)
    - [has\_enum()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_enum)
    - [hasnt\_enum()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_enum)
    - [has\_index()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_index)
    - [hasnt\_index()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_index)
    - [has\_trigger()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_trigger)
    - [hasnt\_trigger()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_trigger)
    - [has\_rule()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_rule)
    - [hasnt\_rule()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_rule)
    - [has\_function()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_function)
    - [hasnt\_function()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_function)
    - [has\_cast()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_cast)
    - [hasnt\_cast()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_cast)
    - [has\_operator()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_operator)
    - [has\_leftop()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_leftop)
    - [has\_rightop()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_rightop)
    - [has\_opclass()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_opclass)
    - [hasnt\_opclass()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_opclass)
    - [has\_role()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_role)
    - [hasnt\_role()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_role)
    - [has\_user()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_user)
    - [hasnt\_user()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_user)
    - [has\_group()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_group)
    - [hasnt\_group()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_group)
    - [has\_language()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_language)
    - [hasnt\_language()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_language)
    - [has\_extension()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_extension)
    - [hasnt\_extension()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_extension)
  + [Mesa para uno](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#tableforone)
    - [has\_column()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_column)
    - [hasnt\_column()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_column)
    - [col\_not\_null()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_not_null)
    - [col\_is\_null()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_is_null)
    - [col\_has\_default()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_has_default)
    - [col\_hasnt\_default()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_hasnt_default)
    - [col\_type\_is()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_type_is)
    - [col\_default\_is()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_default_is)
    - [has\_pk()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_pk)
    - [hasnt\_pk()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_pk)
    - [has\_fk()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_fk)
    - [hasnt\_fk()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#hasnt_fk)
    - [col\_is\_pk()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_is_pk)
    - [col\_isnt\_pk()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_isnt_pk)
    - [col\_is\_fk()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_is_fk)
    - [col\_isnt\_fk()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_isnt_fk)
    - [fk\_ok()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#fk_ok)
    - [has\_unique()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_unique)
    - [col\_is\_unique()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_is_unique)
    - [has\_check()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#has_check)
    - [col\_has\_check()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#col_has_check)
    - [index\_is\_unique()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#index_is_unique)
    - [index\_is\_primary()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#index_is_primary)
    - [is\_partitioned()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_partitioned)
    - [isnt\_partitioned()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isnt_partitioned)
    - [is\_partition\_of()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_partition_of)
    - [is\_clustered()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_clustered)
    - [is\_indexed()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_indexed)
    - [index\_is\_type()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#index_is_type)
  + [Sentirse funky](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#feelingfunky)
    - [can()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#can)
    - [function\_lang\_is()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#function_lang_is)
    - [function\_returns()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#function_returns)
    - [is\_definer()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_definer)
    - [isnt\_definer()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isnt_definer)
    - [is\_strict()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_strict)
    - [isnt\_strict()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isnt_strict)
    - [is\_aggregate()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_aggregate)
    - [isnt\_aggregate()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isnt_aggregate)
    - [volatility\_is()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#volatility_is)
    - [trigger\_is()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#trigger_is)
  + [Base de datos Deets](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#databasedeets)
    - [language\_is\_trusted()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#language_is_trusted)
    - [enum\_has\_labels()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#enum_has_labels)
    - [domain\_type\_is()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#domain_type_is)
    - [domain\_type\_isnt()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#domain_type_isnt)
    - [cast\_context\_is()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#cast_context_is)
    - [is\_superuser()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_superuser)
    - [isnt\_superuser()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#isnt_superuser)
    - [is\_member\_of()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#is_member_of)
    - [rule\_is\_instead()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#rule_is_instead)
    - [rule\_is\_on()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#rule_is_on)
  + [¿Quién me posee?](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#whoownsme)
    - [db\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#db_owner_is)
    - [schema\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#schema_owner_is)
    - [tablespace\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#tablespace_owner_is)
    - [relation\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#relation_owner_is)
    - [table\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#table_owner_is)
    - [view\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#view_owner_is)
    - [materialized\_view\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#materialized_view_owner_is)
    - [sequence\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#sequence_owner_is)
    - [composite\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#composite_owner_is)
    - [foreign\_table\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#foreign_table_owner_is)
    - [index\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#index_owner_is)
    - [function\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#function_owner_is)
    - [language\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#language_owner_is)
    - [opclass\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#opclass_owner_is)
    - [type\_owner\_is ()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#type_owner_is)
  + [Acceso privilegiado](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#privilegedaccess)
    - [database\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#database_privs_are)
    - [tablespace\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#tablespace_privs_are)
    - [schema\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#schema_privs_are)
    - [table\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#table_privs_are)
    - [sequence\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#sequence_privs_are)
    - [any\_column\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#any_column_privs_are)
    - [column\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#column_privs_are)
    - [function\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#function_privs_are)
    - [language\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#language_privs_are)
    - [fdw\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#fdw_privs_are)
    - [server\_privs\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#server_privs_are)
    - [policies\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#policies_are)
    - [policy\_roles\_are()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#policy_roles_are)
    - [policy\_cmd\_is()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#policy_cmd_is)
* [Ninguna prueba para los malvados](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#notestforthewicked)
  + [Diagnósticos](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#diagnostics)
    - [diag()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#diag)
  + [Pruebas condicionales](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#conditionaltests)
    - [skip()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#skip)
    - [todo()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#todo)
    - [todo\_start( why )](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#todo_startwhy)
    - [todo\_start( )](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#todo_start)
    - [todo\_end()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#todo_end)
    - [in\_todo()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#in_todo)
  + [Funciones de utilidad](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#utilityfunctions)
    - [pgtap\_version()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#pgtap_version)
    - [pg\_version()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#pg_version)
    - [pg\_version\_num()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#pg_version_num)
    - [os\_name()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#os_name)
    - [collect\_tap()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#collect_tap)
    - [display\_oper()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#display_oper)
    - [pg\_typeof()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#pg_typeof)
    - [findfuncs()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#findfuncs)
  + [Toque ese lote](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#tapthatbatch)
    - [do\_tap()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#do_tap)
    - [runtests()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#runtests)
* [Secretos de los pgTAP Mavens](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#secretsofthepgtapmavens)
  + [Bucles de estilo relacional](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#relational-styleloops)
* [Componerse](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#composeyourself)
  + [Prueba de funciones de prueba](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#testingtestfunctions)
    - [check\_test()](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#check_test)
* [Compatibilidad](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#compatibility)
  + [10 y hasta](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#10andup)
  + [9.6 y abajo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#9.6anddown)
  + [9.4 y abajo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#9.4anddown)
  + [9.2 y abajo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#9.2anddown)
  + [9.1 y abajo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#9.1anddown)
  + [9.0 y abajo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#9.0anddown)
  + [8.4 y abajo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#8.4anddown)
  + [8.3 y abajo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#8.3anddown)
* [Metadatos](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#metadata)
  + [Repositorio publico](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#publicrepository)
  + [Lista de correo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#maillist)
  + [Autor](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#author)
  + [Creditos](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#credits)
  + [Copyright y licencia](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#copyrightandlicense)

**Sinopsis**

CREATE EXTENSION IF NOT EXISTS pgtap; SELECT plan( 23 ); -- or SELECT \* from no\_plan(); -- Various ways to say "ok" SELECT ok( :have = :want, :test\_description ); SELECT is( :have, :want, :test\_description ); SELECT isnt( :have, :want, :test\_description ); -- Rather than \echo # here's what went wrong SELECT diag( 'here''s what went wrong' ); -- Compare values with LIKE or regular expressions. SELECT alike( :have, :like\_expression, :test\_description ); SELECT unalike( :have, :like\_expression, :test\_description ); SELECT matches( :have, :regex, :test\_description ); SELECT doesnt\_match( :have, :regex, :test\_description ); SELECT cmp\_ok(:have, '=', :want, :test\_description ); -- Skip tests based on runtime conditions. SELECT CASE WHEN :some\_feature THEN collect\_tap( ok( foo(), :test\_description), is( foo(42), 23, :test\_description) ) ELSE skip(:why, :how\_many ) END; -- Mark some tests as to-do tests. SELECT todo(:why, :how\_many); SELECT ok( foo(), :test\_description); SELECT is( foo(42), 23, :test\_description); -- Simple pass/fail. SELECT pass(:test\_description); SELECT fail(:test\_description);

**Instalación**

Para los impacientes, para instalar pgTAP en una base de datos PostgreSQL, simplemente haga esto:

make make install make installcheck

Si encuentra un error como:

"Makefile", line 8: Need an operator

Debe usar GNU make, que puede estar instalado en su sistema como 'gmake':

gmake gmake install gmake installcheck

Si encuentra un error como:

make: pg\_config: Command not found

O:

Makefile:52: \*\*\* pgTAP requires PostgreSQL 8.1 or later. This is . Stop.

Asegúrese de tener pg\_config instalado y en su ruta. Si usó un sistema de administración de paquetes como RPM para instalar PostgreSQL, asegúrese de que el paquete -devel también esté instalado. Si es necesario, dile al proceso de construcción dónde encontrarlo:

env PG\_CONFIG=/path/to/pg\_config make && make install && make installcheck

Y finalmente, si todo eso falla (y si está en PostgreSQL 8.1, probablemente lo hará), copie todo el directorio de distribución al subdirectorio contrib/ del árbol de origen de PostgreSQL y pruébelo sin pg\_config :

env NO\_PGXS=1 make && make install && make installcheck

Si encuentra un error como:

ERROR: must be owner of database regression

Debe ejecutar el conjunto de pruebas con un superusuario, como el superusuario predeterminado "postgres":

make installcheck PGUSER=postgres

**Probando pgTAP con pgTAP**

Además del objetivo de installcheck estándar de installcheck , el objetivo de test utiliza el programa Perl pg\_prove para realizar sus pruebas, que se instalarán con la distribución [TAP :: Parser :: SourceHandler :: pgTAP](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://search.cpan.org/dist/TAP-Parser-SourceHandler-pgTAP&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhg8RZ49R1M8-6t0kegGX_jVrgdmfg) CPAN. Deberá asegurarse de utilizar una base de datos con PL / pgSQL cargado, de lo contrario las pruebas no funcionarán. pg\_prove admite una serie de variables de entorno que podría necesitar usar, incluidas todas las variables de entorno habituales del cliente de PostgreSQL:

* $PGDATABASE
* $PGHOST
* $PGPORT
* $PGUSER

Puede usarlo para ejecutar el conjunto de pruebas como un superusuario de base de datos como:

make test PGUSER=postgres

**Añadiendo pgTAP a una base de datos**

Una vez instalado pgTAP, puede agregarlo a una base de datos. Si está ejecutando PostgreSQL 9.1.0 o superior, es tan simple como conectarse a una base de datos como superusuario y ejecutar:

CREATE EXTENSION IF NOT EXISTS pgtap;

Si ha actualizado su clúster a PostgreSQL 9.1 y ya tiene pgTAP instalado, puede actualizarlo a una extensión correctamente empaquetada con:

CREATE EXTENSION pgtap FROM unpackaged;

Si desea que pgTAP esté disponible para todas las bases de datos nuevas, instálelo en la base de datos "template1":

psql -d template1 -C "CREATE EXTENSION pgtap"

Para desinstalar pgTAP, use DROP EXTENSION :

DROP EXTENSION IF EXISTS pgtap;

Para versiones de PostgreSQL con menos de 9.1.0, deberá ejecutar el script de instalación:

psql -d mydb -f /path/to/pgsql/share/contrib/pgtap.sql

Si desea instalar pgTAP y todos sus objetos de soporte en un esquema específico, use la variable de entorno PGOPTIONS para especificar el esquema, así:

PGOPTIONS=--search\_path=tap psql -d mydb -f pgTAP.sql

Si desea eliminar pgTAP de una base de datos, ejecute el script uninstall\_pgtap.sql :

psql -d dbname -f uninstall\_pgtap.sql

Ambos scripts también se instalarán en el directorio contrib bajo la salida del directorio por pg\_config --sharedir . Así que siempre puedes hacer esto:

psql -d template1 -f `pg\_config --sharedir`/contrib/pgtap.sql

Pero tenga en cuenta que, si ha especificado un esquema utilizando $TAPSCHEMA , ese esquema siempre se creará y las funciones pgTAP se colocarán en él.

**PgTAP Test Scripts**

Puede distribuir pgtap.sql con cualquier distribución de PostgreSQL, como un tipo de datos personalizado. En tal caso, si sus usuarios desean ejecutar su suite de prueba utilizando el test de instalación estándar de installcheck make target, solo asegúrese de establecer variables para mantener las pruebas en silencio, iniciar una transacción, cargar las funciones en su script de prueba y luego deshacer la transacción Al final del guión. Aquí hay un ejemplo:

\unset ECHO \set QUIET 1 -- Turn off echo and keep things quiet. -- Format the output for nice TAP. \pset format unaligned \pset tuples\_only true \pset pager off -- Revert all changes on failure. \set ON\_ERROR\_ROLLBACK 1 \set ON\_ERROR\_STOP true -- Load the TAP functions. BEGIN; \i pgtap.sql -- Plan the tests. SELECT plan(1); -- Run the tests. SELECT pass( 'My test passed, w00t!' ); -- Finish the tests and clean up. SELECT \* FROM finish(); ROLLBACK;

¡Ahora estás listo para ejecutar tu script de prueba!

% psql -d try -Xf test.sql 1..1 ok 1 - My test passed, w00t!

Deberá tener todas esas variables en la secuencia de comandos para asegurarse de que el resultado sea el TAP correcto y que todos los cambios se deshagan, incluida la carga de las funciones de prueba, en el caso de una excepción no detectada.

**Usando pg\_prove**

O ahórrese un poco de esfuerzo, y ejecute un lote de scripts de prueba o todas sus funciones de prueba xUnit al mismo tiempo, mediante el uso de pg\_prove , disponible en la distribución [TAP :: Parser :: SourceHandler :: pgTAP](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://search.cpan.org/dist/TAP-Parser-SourceHandler-pgTAP&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhg8RZ49R1M8-6t0kegGX_jVrgdmfg) CPAN. Si no confía en installcheck , sus scripts de prueba pueden ser mucho menos detallados; no necesita configurar todas las variables adicionales, porque pg\_prove se encarga de eso por usted:

-- Start transaction and plan the tests. BEGIN; SELECT plan(1); -- Run the tests. SELECT pass( 'My test passed, w00t!' ); -- Finish the tests and clean up. SELECT \* FROM finish(); ROLLBACK;

Ahora ejecuta las pruebas. Esto es lo que parece cuando las pruebas pgTAP se ejecutan con pg\_prove :

% pg\_prove -U postgres sql/\*.sql sql/coltap.....ok sql/hastap.....ok sql/moretap....ok sql/pg73.......ok sql/pktap......ok All tests successful. Files=5, Tests=216, 1 wallclock secs ( 0.06 usr 0.02 sys + 0.08 cusr 0.07 csys = 0.23 CPU) Result: PASS

Si está utilizando las pruebas xUnit y solo quiere que pg\_prove ejecute todas a través de la función runtests() , dígale que lo haga:

% pg\_prove -d myapp --runtests

Sí, eso es todo lo que hay que hacer. Llame a pg\_prove --verbose para ver las descripciones de las pruebas individuales, pg\_prove --help para ver otras opciones compatibles, y pg\_prove --man para ver toda la documentación.

**Usando pgTAP**

El propósito de pgTAP es proporcionar una amplia gama de utilidades de prueba que generan TAP. TAP, o el "Protocolo de prueba de cualquier cosa", es un estándar para representar la salida de las pruebas unitarias. Debe su éxito a su formato como una interfaz simple basada en texto que permite un análisis práctico de la máquina y una alta legibilidad para los humanos. TAP comenzó su vida como parte del arnés de prueba para Perl, pero ahora tiene implementaciones en C / C ++, Python, PHP, JavaScript, Perl y, por supuesto, PostgreSQL.

Hay dos formas de usar pgTAP: 1) En scripts de prueba simples que usan un plan para describir las pruebas en el script; o 2) En las funciones de prueba de estilo xUnit que instala en su base de datos y ejecuta todas a la vez en el cliente PostgreSQL de su elección.

**Me encanta cuando un plan viene junto**

Antes que nada, necesitas un plan de pruebas. Básicamente, esto declara cuántas pruebas se ejecutarán en su script para protegerse contra fallas prematuras.

La forma preferida de hacer esto es declarar un plan llamando a la función plan() :

SELECT plan(42);

Hay casos raros en los que no sabrá de antemano cuántas pruebas ejecutará su script. En este caso, puede declarar que no tiene ningún plan. (Trate de evitar usar esto ya que debilita su prueba).

SELECT \* FROM no\_plan();

A menudo, sin embargo, podrás calcular el número de pruebas, de este modo:

SELECT plan( COUNT(\*) ) FROM foo;

Al final de su secuencia de comandos, siempre debe informar a pgTAP que las pruebas se han completado, de modo que pueda emitir cualquier diagnóstico sobre fallas o una discrepancia entre el número planificado de pruebas y el número realmente ejecutado:

SELECT \* FROM finish();

**¡Qué unidad tan dulce!**

Si está acostumbrado a los marcos de prueba de xUnit, puede recopilar todas las pruebas en funciones de base de datos y ejecutarlas todas a la vez con runtests() . Esto es similar a cómo funciona [PGUnit](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://en.dklab.ru/lib/dklab_pgunit/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhglXXvDkRMYnMXlpeIthrLEjj4KWg) . La función runtests() hace todo el trabajo de encontrar y ejecutar sus funciones de prueba en transacciones individuales. Incluso es compatible con las funciones de configuración y desmontaje. Para usarlo, escriba las funciones de prueba de unidad para que devuelvan un conjunto de resultados de texto y luego use las funciones de aseveración pgTAP para devolver los valores TAP. Aquí hay un ejemplo, probando una tabla de users hipotéticos:

CREATE OR REPLACE FUNCTION setup\_insert( ) RETURNS SETOF TEXT AS $$ BEGIN RETURN NEXT is( MAX(nick), NULL, 'Should have no users') FROM users; INSERT INTO users (nick) VALUES ('theory'); END; $$ LANGUAGE plpgsql; CREATE OR REPLACE FUNCTION test\_user( ) RETURNS SETOF TEXT AS $$ SELECT is( nick, 'theory', 'Should have nick') FROM users; END; $$ LANGUAGE sql;

Consulte a continuación para obtener detalles sobre las funciones de afirmación pgTAP. Una vez que haya definido sus funciones de prueba de unidad, puede ejecutar sus pruebas en cualquier momento utilizando la función runtests() :

SELECT \* FROM runtests();

Cada función de prueba se ejecutará dentro de su propia transacción y se retrotraerá cuando la función se complete (o después de que se hayan ejecutado las funciones de desmontaje). Los resultados del TAP serán enviados a su cliente.

**Descripciones de Pruebas**

Por convención, a cada prueba se le asigna un número en orden. Esto se hace en gran parte automáticamente para usted. Sin embargo, a menudo es muy útil describir cada prueba. ¿Prefieres ver esto?

ok 4 not ok 5 ok 6

¿O esto?

ok 4 - basic multi-variable not ok 5 - simple exponential ok 6 - force == mass \* acceleration

Este último te da una idea de lo que falló. También hace que sea más fácil encontrar la prueba en su script, simplemente busque "exponencial simple".

Todas las funciones de prueba toman un argumento de descripción. Es opcional, pero es altamente recomendable que lo uses.

**xunidad nombres de funciones**

A veces es útil extraer los nombres de las funciones de prueba xUnit de la salida TAP, especialmente cuando se usa el estilo xUnit con Continuous Integration Server como Hudson o TeamCity. De forma predeterminada, pgTAP muestra estos nombres como comentarios, pero puede cambiar este comportamiento anulando la función diag\_test\_name . Por ejemplo:

CREATE OR REPLACE FUNCTION diag\_test\_name(TEXT) RETURNS TEXT AS $$ SELECT diag('test: ' || $1 ); $$ LANGUAGE SQL;

Esto mostrará

# test: my\_example\_test\_function\_name

en lugar de

# my\_example\_test\_function\_name()

Esto simplifica los nombres de las pruebas de análisis de los comentarios de TAP.

**Estoy bien, no estás bien**

El propósito básico de pgTAP, y de cualquier marco de prueba de emisión de TAP, en realidad, es imprimir "ok #" o "no ok #", dependiendo de si una prueba determinada tuvo éxito o no. Todo lo demás es simplemente salsa.

Todas las siguientes funciones devuelven "ok" o "no ok" dependiendo de si la prueba tuvo éxito o no.

**ok()**

SELECT ok( :boolean, :description ); SELECT ok( :boolean );

**Parámetros**

:boolean

Un valor booleano que indica éxito o fracaso.

:description

Una breve descripción de la prueba.

Esta función simplemente evalúa cualquier expresión booleana y la utiliza para determinar si la prueba tuvo éxito o no. Una expresión verdadera pasa, una falsa falla. Muy simple.

Por ejemplo:

SELECT ok( 9 ^ 2 = 81, 'simple exponential' ); SELECT ok( 9 < 10, 'simple comparison' ); SELECT ok( 'foo' ~ '^f', 'simple regex' ); SELECT ok( active = true, name || widget active' ) FROM widgets;

(Mnemónico: "Esto está bien".)

La :description es una :description muy breve de la prueba que se imprimirá. Es muy fácil encontrar una prueba en su script cuando falla y le da a otros una idea de sus intenciones. La descripción es opcional, pero recomendamos encarecidamente su uso.

Si un ok() falla, producirá algunos diagnósticos:

not ok 18 - sufficient mucus # Failed test 18: "sufficient mucus"

Además, si el argumento del resultado de la prueba booleana se pasa como NULL lugar de true o false , ok() supondrá un error en la prueba y adjuntará un diagnóstico adicional:

not ok 18 - sufficient mucus # Failed test 18: "sufficient mucus" # (test result was NULL)

**is()**

**isnt()**

SELECT is( :have, :want, :description ); SELECT is( :have, :want ); SELECT isnt( :have, :want, :description ); SELECT isnt( :have, :want );

**Parámetros**

:have

Valor a prueba.

:want

Valor que :have se espera que tenga. Debe ser el mismo tipo de datos.

:description

Una breve descripción de la prueba.

Similar a ok() , is() e isnt() comparan sus dos argumentos con IS NOT DISTINCT FROM ( = ) Y IS DISTINCT FROM ( <> ) respectivamente y usa el resultado para determinar si la prueba tuvo éxito o no. Así que estos:

-- Is the ultimate answer 42? SELECT is( ultimate\_answer(), 42, 'Meaning of Life' ); -- foo() doesn't return empty SELECT isnt( foo(), '', 'Got some foo' );

son similares a estos:

SELECT ok( ultimate\_answer() = 42, 'Meaning of Life' ); SELECT ok( foo() <> '', 'Got some foo' );

(Mnemónico: "Esto es eso". "Esto no es eso".)

*Nota:* Gracias al uso de la construcción IS [ NOT ] DISTINCT FROM , NULL s no se tratan como incógnitas por is() o isnt() . Es decir, si :have y :want son NULL , la prueba pasará, y si solo uno de ellos es NULL , la prueba fallará.

Entonces, ¿por qué usar estas funciones de prueba? Producen mejores diagnósticos en caso de fallo. ok() no puede saber para qué está probando (más allá de la descripción), pero is() y isnt() saber qué prueba fue y por qué falló. Por ejemplo esta prueba:

\set foo '\'waffle\'' \set bar '\'yarblokos\'' SELECT is( :foo::text, :bar::text, 'Is foo the same as bar?' );

Producirá algo como esto:

# Failed test 17: "Is foo the same as bar?" # have: waffle # want: yarblokos

Para que pueda averiguar qué salió mal sin volver a ejecutar la prueba.

Se recomienda que use is() y isnt() sobre ok() cuando sea posible. Incluso puede usarlos para comparar registros en PostgreSQL 8.4 y posteriores:

SELECT is( users.\*, ROW(1, 'theory', true)::users ) FROM users WHERE nick = 'theory';

**matches()**

SELECT matches( :have, :regex, :description ); SELECT matches( :have, :regex );

**Parámetros**

:have

Valor para igualar.

:regex

Una expresión regular.

:description

Una breve descripción de la prueba.

Similar a ok() , matches() match :have against the regex :regex .

Así que esto:

SELECT matches( :this, '^that', 'this is like that' );

es parecido a:

SELECT ok( :this ~ '^that', 'this is like that' );

(Mnemónico "Esto coincide con eso".)

Sus ventajas sobre ok() son similares a las de is() y isnt() : mejores diagnósticos en caso de fallo.

**imatches()**

SELECT imatches( :have, :regex, :description ); SELECT imatches( :have, :regex );

**Parámetros**

:have

Valor para igualar.

:regex

Una expresión regular.

:description

Una breve descripción de la prueba.

Igual que matches() excepto que la expresión regular se compara con :have mayúsculas y minúsculas.

**doesnt\_match()**

**doesnt\_imatch()**

SELECT doesnt\_match( :have, :regex, :description ); SELECT doesnt\_match( :have, :regex ); SELECT doesnt\_imatch( :have, :regex, :description ); SELECT doesnt\_imatch( :have, :regex );

**Parámetros**

:have

Valor para igualar.

:regex

Una expresión regular.

:description

Una breve descripción de la prueba.

Estas funciones funcionan exactamente como lo hacen matches() y imatches() , solo que verifican si :have *no* coincide con el patrón dado.

**alike()**

**ialike()**

SELECT alike( :this, :like, :description ); SELECT alike( :this, :like ); SELECT ialike( :this, :like, :description ); SELECT ialike( :this, :like );

**Parámetros**

:have

Valor para igualar.

:like

Un patrón SQL LIKE .

:description

Una breve descripción de la prueba.

Similar a matches() , alike() match :hve contra el patrón SQL LIKE :like . ialike() coincide con mayúsculas y minúsculas.

Así que esto:

SELECT ialike( :have, 'that%', 'this is alike that' );

es parecido a:

SELECT ok( :have ILIKE 'that%', 'this is like that' );

(Mnemónico "Esto es así".)

Sus ventajas sobre ok() son similares a las de is() y isnt() : mejores diagnósticos en caso de fallo.

**unalike()**

**unialike()**

SELECT unalike( :this, :like, :description ); SELECT unalike( :this, :like ); SELECT unialike( :this, :like, :description ); SELECT unialike( :this, :like );

**Parámetros**

:have

Valor para igualar.

:like

Un patrón SQL LIKE .

:description

Una breve descripción de la prueba.

Funciona exactamente alike() , solo verifica si :have *no* coincide con el patrón dado.

**cmp\_ok()**

SELECT cmp\_ok( :have, :op, :want, :description ); SELECT cmp\_ok( :have, :op, :want );

**Parámetros**

:have

Valor para comparar.

:op

Un operador de SQL especificado como una cadena.

:want

Valor para comparar con :have usando el operador :op .

:description

Una breve descripción de la prueba.

A mitad de camino entre ok() y is() encuentra cmp\_ok() . Esta función le permite comparar dos argumentos utilizando cualquier operador binario.

-- ok( :have = :want ); SELECT cmp\_ok( :have, '=', :want, 'this = that' ); -- ok( :have >= :want ); SELECT cmp\_ok( :have, '>=', :want, 'this >= that' ); -- ok( :have && :want ); SELECT cmp\_ok( :have, '&&', :want, 'this && that' );

Su ventaja sobre ok() es que cuando la prueba falla, sabrás qué :have y :want fueron:

not ok 1 # Failed test 1: # '23' # && # NULL

Tenga en cuenta que si el valor devuelto por la operación es NULL , se considerará que la prueba ha fallado. Esto puede no ser lo que esperabas si tu prueba fuera, por ejemplo:

SELECT cmp\_ok( NULL, '=', NULL );

Pero en ese caso, probablemente deberías usar is() , en su lugar.

**pass()**

**fail()**

SELECT pass( :description ); SELECT pass( ); SELECT fail( :description ); SELECT fail( );

**Parámetros**

:description

Una breve descripción de la prueba.

A veces solo quieres decir que las pruebas han pasado. Por lo general, el caso es que tiene una condición complicada que es difícil de encajar en un ok() . En este caso, simplemente puede usar pass() (para declarar que la prueba es correcta) o fail() (porque no está bien). Son sinónimos de ok(1) y ok(0) .

Usa estas funciones muy, muy, muy escasamente.

**isa\_ok()**

SELECT isa\_ok( :have, :regtype, :name ); SELECT isa\_ok( :have, :regtype );

**Parámetros**

:have

Valor para comprobar el tipo de.

:regtype

Nombre de un tipo de datos SQL.

:name

Un nombre para el valor que se compara.

Comprueba si el valor dado es de un tipo particular. La descripción y los diagnósticos de esta prueba normalmente solo se refieren al "valor". Si desea que sean más específicos, puede proporcionar un :name . Por ejemplo, podría decir "el valor de retorno" cuando esté examinando el resultado de una llamada de función:

SELECT isa\_ok( length('foo'), 'integer', 'The return value from length()' );

En cuyo caso, la descripción será "El valor de retorno de length () es un entero".

En el caso de una falla, el mensaje de diagnóstico le dirá qué tipo de valor es en realidad:

not ok 12 - the value isa integer[] # the value isn't a "integer[]" it's a "boolean"

**Persiguiendo tu consulta**

A veces, solo tienes que probar una consulta. Me refiero a los resultados de una consulta completa, no solo a las funciones de aserción escalar que hemos visto hasta ahora. pgTAP proporciona una serie de funciones para ayudarlo a probar sus consultas, cada una de las cuales toma una o dos declaraciones SQL como argumentos. Por ejemplo:

SELECT throws\_ok('SELECT divide\_by(0)');

Sí, como cuerdas. Por supuesto, a menudo tendrá que hacer algo complejo en su SQL, y citar SQL en cadenas en lo que es, después de todo, una aplicación SQL, es un PITA innecesario. Cada una de las funciones de ejecución de consultas en esta sección, por lo tanto, es compatible con una alternativa para hacer que sus pruebas sean más sencillas: utilizando declaraciones preparadas.

[Las declaraciones preparadas le](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.postgresql.org/docs/current/static/sql-prepare.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhhZarQJF-pQ7AW8DN9cPqYYcbuOGg) permiten escribir SQL y simplemente pasar los nombres de las declaraciones preparadas para probar funciones. Por ejemplo, el ejemplo anterior se puede reescribir como:

PREPARE mythrow AS SELECT divide\_by(0); SELECT throws\_ok('mythrow');

pgTAP asume que un argumento de SQL sin caracteres de espacio o que comience con un carácter de comillas dobles es una declaración preparada y simplemente EXECUTE . Si necesita pasar argumentos a una declaración preparada, tal vez porque planea usarlo en varias pruebas para devolver valores diferentes, simplemente EXECUTE usted mismo. Aquí hay un ejemplo con una declaración preparada con un espacio en su nombre, y uno donde se deben pasar los argumentos:

PREPARE "my test" AS SELECT \* FROM active\_users() WHERE name LIKE 'A%'; PREPARE expect AS SELECT \* FROM users WHERE active = $1 AND name LIKE $2; SELECT results\_eq( '"my test"', 'EXECUTE expect( true, ''A%'' )' );

Dado que "mi prueba" se declaró con comillas dobles, se debe pasar con comillas dobles. Y dado que la llamada a "esperar" incluía espacios (para mantenerlo legible), se requería la palabra clave EXECUTE .

También puede utilizar una instrucción VALUES , tanto en la cadena de consulta como en una declaración preparada. Un ejemplo inútil:

PREPARE myvals AS VALUES (1, 2), (3, 4); SELECT set\_eq( 'myvals', 'VALUES (1, 2), (3, 4)' );

Aquí hay una ventaja si necesita verificar los resultados de una consulta que devuelve una sola columna: para aquellas funciones que toman dos argumentos de consulta, la segunda puede ser una matriz. Echale un vistazo:

SELECT results\_eq( 'SELECT \* FROM active\_user\_ids()', ARRAY[ 2, 3, 4, 5] );

La primera consulta *debe* devolver solo una columna del mismo tipo que los valores de la matriz. Si necesita probar más columnas, deberá usar dos consultas.

Teniendo en cuenta estas técnicas, sigue leyendo para conocer toda la bondad de las pruebas de consulta.

**El error es humano**

A veces, solo desea saber que una consulta en particular generará un error. O quizás desee asegurarse de que una consulta *no* genere un error. Para tales casos, proporcionamos un par de funciones de prueba para asegurarnos de que sus consultas sean tan propensas a errores como usted cree que deberían ser.

**throws\_ok()**

SELECT throws\_ok( :sql, :errcode, :ermsg, :description ); SELECT throws\_ok( :sql, :errcode, :ermsg ); SELECT throws\_ok( :sql, :errcode ); SELECT throws\_ok( :sql, :errmsg, :description ); SELECT throws\_ok( :sql, :errmsg ); SELECT throws\_ok( :sql );

**Parámetros**

:sql

Una declaración SQL o el nombre de una declaración preparada, que se pasa como una cadena.

:errcode

Un [código de error de PostgreSQL](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.postgresql.org/docs/current/static/errcodes-appendix.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhi99aTIQW2bo7nwu4IyN5kXPxIvlg)

:errmsg

Un mensaje de error.

:description

Una breve descripción de la prueba.

Cuando quiera asegurarse de que PostgreSQL lanza una excepción, use throws\_ok() para probarlo.

El primer argumento debe ser el nombre de una declaración preparada o, si no, una cadena que representa la consulta que se ejecutará (consulte el [resumen](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Pursuing+Your+Query) para obtener detalles sobre los argumentos de la consulta). throws\_ok() usará la sentencia PL / pgSQL EXECUTE para ejecutar la consulta y detectar cualquier excepción.

El segundo argumento debe ser un código de error de excepción, que es una cadena de cinco caracteres (si se trata de números que solo consisten en números y los pasa como un entero, seguirá funcionando). Si este valor no es NULL , throws\_ok() comprobará la excepción lanzada para asegurarse de que sea la excepción esperada. Para obtener una lista completa de los códigos de error, consulte el [Apéndice A.](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.postgresql.org/docs/current/static/errcodes-appendix.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhi99aTIQW2bo7nwu4IyN5kXPxIvlg) en la [documentación de PostgreSQL](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.postgresql.org/docs/current/static/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhiEqpGh_R5fX4RKPsk_PmNWvzpdOg) .

El tercer argumento es un mensaje de error. Esto será más útil para las funciones que ha escrito que generan excepciones, para que pueda probar el mensaje de excepción que ha lanzado. De lo contrario, para los errores principales, deberá tener cuidado con los mensajes de error localizados. Un truco para evitar los mensajes de error localizados es pasar NULL como el tercer argumento. Esto le permite pasar una descripción como el cuarto argumento.

El cuarto argumento es, por supuesto, una breve descripción de la prueba. Aquí hay un ejemplo útil:

PREPARE my\_thrower AS INSERT INTO try (id) VALUES (1); SELECT throws\_ok( 'my\_thrower', '23505', 'duplicate key value violates unique constraint "try\_pkey"', 'We should get a unique violation for a duplicate PK' );

Para las formas de dos y tres argumentos de throws\_ok() , si el segundo argumento tiene exactamente cinco bytes de longitud, se supone que es un código de error y el tercer argumento opcional es el mensaje de error. De lo contrario, se supone que el segundo argumento es un mensaje de error y el tercer argumento es una descripción. Si, por algún motivo, necesita probar un mensaje de error de cinco bytes de longitud, use el formulario de cuatro argumentos.

Una throws\_ok() fallida throws\_ok() produce un mensaje de diagnóstico apropiado. Por ejemplo:

# Failed test 81: "This should die a glorious death" # caught: 23505: duplicate key value violates unique constraint "try\_pkey" # wanted: 23502: null value in column "id" violates not-null constraint

Idea prestada del módulo Test :: Exception Perl.

**throws\_like()**

**throws\_ilike()**

SELECT throws\_like( :sql, :like, :description ); SELECT throws\_like( :sql, :like ); SELECT throws\_ilike( :sql, :like, :description ); SELECT throws\_ilike( :sql, :like );

**Parámetros**

:sql

Una declaración SQL o el nombre de una declaración preparada, que se pasa como una cadena.

:like

Un patrón SQL LIKE .

:description

Una breve descripción de la prueba.

Me gusta throws\_ok() , pero prueba que un mensaje de error de excepción coincide con un patrón LIKE SQL. La variante throws\_ilike() coincide con mayúsculas y minúsculas. Un ejemplo:

PREPARE my\_thrower AS INSERT INTO try (tz) VALUES ('America/Moscow'); SELECT throws\_like( 'my\_thrower', '%"timezone\_check"', 'We should error for invalid time zone' );

Una throws\_like() fallida de throws\_like() produce un mensaje de diagnóstico apropiado. Por ejemplo:

# Failed test 85: "We should error for invalid time zone" # error message: 'value for domain timezone violates check constraint "tz\_check"' # doesn't match: '%"timezone\_check"'

**throws\_matching()**

**throws\_imatching()**

SELECT throws\_matching( :sql, :regex, :description ); SELECT throws\_matching( :sql, :regex ); SELECT throws\_imatching( :sql, :regex, :description ); SELECT throws\_imatching( :sql, :regex );

**Parámetros**

:sql

Una declaración SQL o el nombre de una declaración preparada, que se pasa como una cadena.

:regex

Una expresión regular.

:description

Una breve descripción de la prueba.

Me gusta throws\_ok() , pero prueba que un mensaje de error de excepción coincide con una expresión regular. La variante throws\_imatching() coincide con mayúsculas y minúsculas. Un ejemplo:

PREPARE my\_thrower AS INSERT INTO try (tz) VALUES ('America/Moscow'); SELECT throws\_matching( 'my\_thrower', '.+"timezone\_check"', 'We should error for invalid time zone' );

Una throws\_matching() fallida throws\_matching() produce un mensaje de diagnóstico apropiado. Por ejemplo:

# Failed test 85: "We should error for invalid time zone" # error message: 'value for domain timezone violates check constraint "tz\_check"' # doesn't match: '.+"timezone\_check"'

**lives\_ok()**

SELECT lives\_ok( :sql, :description ); SELECT lives\_ok( :sql );

**Parámetros**

:sql

Una declaración SQL o el nombre de una declaración preparada, que se pasa como una cadena.

:description

Una breve descripción de la prueba.

El inverso de throws\_ok() , lives\_ok() asegura que una declaración SQL *no* arroja una excepción. Pase el nombre de una declaración preparada o una cadena de código SQL (consulte el [resumen](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Pursuing+Your+Query) para obtener detalles de los argumentos de consulta). El segundo argumento opcional es la descripción de la prueba. Un ejemplo:

SELECT lives\_ok( 'INSERT INTO try (id) VALUES (1)', 'We should not get a unique violation for a new PK' );

Una lives\_ok() fallida de lives\_ok() produce un mensaje de diagnóstico apropiado. Por ejemplo:

# Failed test 85: "don't die, little buddy!" # died: 23505: duplicate key value violates unique constraint "try\_pkey"

Idea prestada del módulo Test :: Exception Perl.

**performs\_ok()**

SELECT performs\_ok( :sql, :milliseconds, :description ); SELECT performs\_ok( :sql, :milliseconds );

**Parámetros**

:sql

Una declaración SQL o el nombre de una declaración preparada, que se pasa como una cadena.

:milliseconds

Número de milisegundos.

:description

Una breve descripción de la prueba.

Esta función se asegura de que una sentencia de SQL funcione bien. Lo hace sincronizando su ejecución y fallando si la ejecución toma más tiempo que el número especificado de milisegundos. Un ejemplo:

PREPARE fast\_query AS SELECT id FROM try WHERE name = 'Larry'; SELECT performs\_ok( 'fast\_query', 250, 'A select by name should be fast' );

El primer argumento debe ser el nombre de una declaración preparada o una cadena que represente la consulta que se ejecutará (consulte el [resumen](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Pursuing+Your+Query) para obtener detalles sobre los argumentos de la consulta). performs\_ok() usará la sentencia PL / pgSQL EXECUTE para ejecutar la consulta.

El segundo argumento es el número máximo de milisegundos que debe tomar para que se ejecute la instrucción SQL. Este argumento es numérico, por lo que incluso puedes usar fracciones de milisegundos si flota tu barco.

El tercer argumento es la descripción habitual. Si no se proporciona, performs\_ok() generará la descripción "Debería ejecutarse en menos de $ milisegundos ms". Es probable que desee proporcionar su propia descripción si tiene más de un par de estos en un script o función de prueba.

Si una prueba performs\_ok() falla, produce los mensajes de diagnóstico apropiados. Por ejemplo:

# Failed test 19: "The lookup should be fast!" # runtime: 200.266 ms # exceeds: 200 ms

*Nota:* Hay un poco de tiempo adicional incluido en el tiempo de ejecución para la sobrecarga de EXECUTE de PL / pgSQL, que debe compilar y ejecutar la cadena SQL. Usted querrá tener en cuenta esto y rellenar sus estimaciones en consecuencia. Es mejor pensar en esto como una comparación de fuerza bruta de los tiempos de ejecución, para garantizar que la consulta no sea *realmente* lenta (piense en segundos).

**performs\_within()**

SELECT performs\_within( :sql, :average\_milliseconds, :within, :iterations, :description ); SELECT performs\_within( :sql, :average\_milliseconds, :within, :description ); SELECT performs\_within( :sql, :average\_milliseconds, :within, :iterations); SELECT performs\_within( :sql, :average\_milliseconds, :within);

**Parámetros**

:sql

Una declaración SQL o el nombre de una declaración preparada, que se pasa como una cadena.

:average\_milliseconds

Número de milisegundos que la consulta debe tomar en promedio.

:within

La cantidad de milisegundos que el promedio puede variar.

:iterations

El número de veces para ejecutar la consulta.

:description

Una breve descripción de la prueba.

Esta función asegura que una declaración SQL, en promedio, se ejecute dentro de una ventana esperada. Lo hace ejecutando la consulta un valor predeterminado de 10 veces. Expulsa el 10% superior e inferior de las carreras y promedia el 80% medio de las carreras que realizó. Si el tiempo de ejecución promedio está fuera del rango especificado por within , la prueba fallará. Un ejemplo:

PREPARE fast\_query AS SELECT id FROM try WHERE name = 'Larry'; SELECT performs\_within( 'fast\_query', 250, 10, 100, 'A select by name should be fast' );

El primer argumento debe ser el nombre de una declaración preparada o una cadena que represente la consulta que se ejecutará (consulte el [resumen](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Pursuing+Your+Query) para obtener detalles sobre los argumentos de la consulta). performs\_within() usará la sentencia PL / pgSQL EXECUTE para ejecutar la consulta.

El segundo argumento es el número promedio de milisegundos que debe tomar para que se ejecute la instrucción SQL. Este argumento es numérico, por lo que incluso puedes usar fracciones de milisegundos si flota tu barco.

El tercer argumento es la cantidad de milisegundos que se permite que la consulta varíe alrededor del promedio y aún así pase la prueba. Si el promedio de la consulta cae fuera de esta ventana, ya sea demasiado rápido o demasiado lento, fallará.

El cuarto argumento es el número de iteraciones o la descripción habitual. Si no se proporciona, performs\_within() ejecutará 10 ejecuciones de la consulta y generará la descripción "Debe ejecutarse en $ average\_milliseconds +/- $ dentro de ms". Es probable que desee proporcionar su propia descripción si tiene más de un par de estos en un script o función de prueba.

El quinto argumento es la descripción habitual como se describe anteriormente, asumiendo que también ha especificado el número de iteraciones.

Si una prueba performs\_within() falla, produce los mensajes de diagnóstico apropiados. Por ejemplo:

# Failed test 19: "The lookup should be fast!" # average runtime: 210.266 ms # desired average: 200 +/- 10 ms

*Nota:* Hay un poco de tiempo adicional incluido en el tiempo de ejecución para la sobrecarga de EXECUTE de PL / pgSQL, que debe compilar y ejecutar la cadena SQL. Usted querrá tener en cuenta esto y rellenar sus estimaciones en consecuencia. Es mejor pensar en esto como una comparación de fuerza bruta de los tiempos de ejecución, para garantizar que la consulta no sea *realmente* lenta (piense en segundos).

**¿Te identificas?**

Así que tienes tus funciones básicas de comparación escalar, ¿qué pasa con las relaciones? ¿Es posible que tenga algunas declaraciones SELECT bastante peludas en vistas o funciones para probar? Tenemos sus funciones de prueba de relaciones aquí.

**results\_eq()**

SELECT results\_eq( :sql, :sql, :description ); SELECT results\_eq( :sql, :sql ); SELECT results\_eq( :sql, :array, :description ); SELECT results\_eq( :sql, :array ); SELECT results\_eq( :cursor, :cursor, :description ); SELECT results\_eq( :cursor, :cursor ); SELECT results\_eq( :sql, :cursor, :description ); SELECT results\_eq( :sql, :cursor ); SELECT results\_eq( :cursor, :sql, :description ); SELECT results\_eq( :cursor, :sql ); SELECT results\_eq( :cursor, :array, :description ); SELECT results\_eq( :cursor, :array );

**Parámetros**

:sql

Una declaración SQL o el nombre de una declaración preparada, que se pasa como una cadena.

:array

Una matriz de valores que representan los valores de una fila de una sola columna.

:cursor

Un valor de refcursor PostgreSQL que representa un cursor nombrado.

:description

Una breve descripción de la prueba.

Hay tres formas de probar conjuntos de resultados en pgTAP. Quizás lo más intuitivo es hacer una comparación directa de resultados fila por fila para garantizar que sean exactamente lo que usted espera, en el orden que espera. Casualmente, esto es exactamente cómo se comporta results\_eq() . Así es como lo usa: simplemente pase dos instrucciones SQL o nombres de instrucciones preparadas (o alguna combinación; consulte el [resumen](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Pursuing+Your+Query) para obtener detalles de los argumentos de consulta) y una descripción opcional.

Por ejemplo, supongamos que tiene una función, active\_users() , que devuelve un conjunto de filas de la tabla de usuarios. Para asegurarse de que devuelve las filas que espera, puede hacer algo como esto:

SELECT results\_eq( 'SELECT \* FROM active\_users()', 'SELECT \* FROM users WHERE active', 'active\_users() should return active users' );

Sugerencia: si desea codificar los valores para comparar, use una declaración VALUES lugar de una consulta, como:

SELECT results\_eq( 'SELECT \* FROM active\_users()', $$VALUES ( 42, 'Anna'), (19, 'Strongrrl'), (39, 'Theory')$$, 'active\_users() should return active users' );

Si los resultados devueltos por el primer argumento consisten en una sola columna, el segundo argumento puede ser una matriz:

SELECT results\_eq( 'SELECT \* FROM active\_user\_ids()', ARRAY[ 2, 3, 4, 5] );

En general, se recomienda encarecidamente el uso de declaraciones preparadas para mantener su código de prueba en SQLish (incluso puede usar VALUES en declaraciones preparadas).Pero tenga en cuenta que, dado que results\_eq()hace una comparación fila por fila, los resultados de los dos argumentos de consulta deben estar exactamente en el mismo orden, con exactamente los mismos tipos de datos, para poder pasar. En términos prácticos, significa que debe asegurarse de que sus resultados nunca se ordenan de forma inequívoca.

Por ejemplo, supongamos que desea comparar consultas con una personstabla. La forma más sencilla de ordenar es por name, como en:

try=# select \* from people order by name; name | age --------+----- Damian | 19 Larry | 53 Tom | 44 Tom | 35 (4 rows)

Pero una ejecución diferente de la misma consulta podría tener las filas en orden diferente:

try=# select \* from people order by name; name | age --------+----- Damian | 19 Larry | 53 Tom | 35 Tom | 44 (4 rows)

Observe cómo se invierten las dos filas "Tom". El resultado es que debe asegurarse de que sus consultas estén siempre ordenadas. En un caso como el anterior, significa clasificar tanto en la namecolumna como en la agecolumna. Si el orden de clasificación de sus resultados no es importante, considere usar set\_eq()o en su bag\_eq()lugar.

Internamente, results\_eq()convierte sus sentencias de SQL en cursores para que pueda iterar sobre ellas una fila a la vez. Convenientemente, este comportamiento también está directamente disponible para usted. En lugar de pasar de alguna instrucción SQL arbitrario o el nombre de una declaración preparada, basta con crear un cursor y pasar *que* en, así:

DECLARE cwant CURSOR FOR SELECT \* FROM active\_users(); DECLARE chave CURSOR FOR SELECT \* FROM users WHERE active ORDER BY name; SELECT results\_eq( 'cwant'::refcursor, 'chave'::refcursor, 'Gotta have those active users!' );

La clave es asegurar que los nombres de los cursores se pasan como refcursors. Esto permite results\_eq()desambiguarlos de las declaraciones preparadas. Y, por supuesto, puede mezclar y unir cursores, declaraciones preparadas y SQL tanto como desee. Aquí hay un ejemplo que usa una declaración preparada y un cursor (restablecer) para los resultados esperados:

PREPARE users\_test AS SELECT \* FROM active\_users(); MOVE BACKWARD ALL IN chave; SELECT results\_eq( 'users\_test', 'chave'::refcursor, 'Gotta have those active users!' );

Independientemente de los tipos de argumentos que pase, en el caso de una falla de prueba, results\_eq()le ofrecerá un buen mensaje de diagnóstico para decirle en qué fila difieren los resultados, algo así como:

# Failed test 146 # Results differ beginning at row 3: # have: (1,Anna) # want: (22,Betty)

Si hay diferentes números de filas en cada conjunto de resultados, una fila no existente se representará como "NULL":

# Failed test 147 # Results differ beginning at row 5: # have: (1,Anna) # want: NULL

En PostgreSQL 8.4 o superior, si el número de columnas varía entre los conjuntos de resultados, o si los resultados son de diferentes tipos de datos, obtendrá diagnósticos así:

# Failed test 148 # Number of columns or their types differ between the queries: # have: (1) # want: (foo,1)

En PostgreSQL 8.3 y hacia abajo, las filas se convierten en texto para comparar, en lugar de compararlas como recordobjetos. El inconveniente de esta necesidad es que la prueba no puede detectar incompatibilidades en los números o tipos de columna, o diferencias en las columnas que se convierten en la misma representación de texto. Por ejemplo, una NULLcolumna será equivalente a una cadena vacía. Como resultado, pgTAP no mostrará los valores havey wantsi son iguales, solo el mensaje de error, así:

# Failed test 149 # Number of columns or their types differ between the queries

**results\_ne()**

SELECT results\_ne( :sql, :sql, :description ); SELECT results\_ne( :sql, :sql ); SELECT results\_ne( :sql, :array, :description ); SELECT results\_ne( :sql, :array ); SELECT results\_ne( :cursor, :cursor, :description ); SELECT results\_ne( :cursor, :cursor ); SELECT results\_ne( :sql, :cursor, :description ); SELECT results\_ne( :sql, :cursor ); SELECT results\_ne( :cursor, :sql, :description ); SELECT results\_ne( :cursor, :sql ); SELECT results\_ne( :cursor, :array, :description ); SELECT results\_ne( :cursor, :array );

**Parámetros**

:sql

Una declaración SQL o el nombre de una declaración preparada, que se pasa como una cadena.

:array

Una matriz de valores que representan los valores de una fila de una sola columna.

:cursor

Un refcursorvalor de PostgreSQL que representa un cursor nombrado.

:description

Una breve descripción de la prueba.

The inverse of results\_eq() , this function tests that query results are not equivalent. Note that, like results\_ne() , order matters, so you can actually have the same sets of results in the two query arguments and the test will pass if they're merely in a different order. More than likely what you really want is results\_eq() or set\_ne() . But this function is included for completeness and is kind of cute, so enjoy. If a results\_ne() test fails, however, there will be no diagnostics, because, well, the results will be the same!

Note that the caveats for results\_ne() on PostgreSQL 8.3 and down apply to results\_ne() as well.

**set\_eq()**

SELECT set\_eq( :sql, :sql, :description ); SELECT set\_eq( :sql, :sql ); SELECT set\_eq( :sql, :array, :description ); SELECT set\_eq( :sql, :array );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:array

An array of values representing a single-column row values.

:description

A short description of the test.

Sometimes you don't care what order query results are in, or if there are duplicates. In those cases, use set\_eq() to do a simple set comparison of your result sets. As long as both queries return the same records, regardless of duplicates or ordering, a set\_eq() test will pass.

The SQL arguments can be the names of prepared statements or strings containing an SQL query (see the [summary](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Pursuing+Your+Query) for query argument details), or even one of each. If the results returned by the first argument consist of a single column, the second argument may be an array:

SELECT set\_eq( 'SELECT \* FROM active\_user\_ids()', ARRAY[ 2, 3, 4, 5] );

In whatever case you choose to pass arguments, a failing test will yield useful diagnostics, such as:

# Failed test 146 # Extra records: # (87,Jackson) # (1,Jacob) # Missing records: # (44,Anna) # (86,Angelina)

In the event that you somehow pass queries that return rows with different types of columns, pgTAP will tell you that, too:

# Failed test 147 # Columns differ between queries: # have: (integer,text) # want: (text,integer)

This of course extends to sets with different numbers of columns:

# Failed test 148 # Columns differ between queries: # have: (integer) # want: (text,integer)

**set\_ne()**

SELECT set\_ne( :sql, :sql, :description ); SELECT set\_ne( :sql, :sql ); SELECT set\_ne( :sql, :array, :description ); SELECT set\_ne( :sql, :array );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:array

An array of values representing a single-column row values.

:description

A short description of the test.

The inverse of set\_eq() , this function tests that the results of two queries are *not* the same. The two queries can as usual be the names of prepared statements or strings containing an SQL query (see the [summary](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Pursuing+Your+Query) for query argument details), or even one of each. The two queries, however, must return results that are directly comparable — that is, with the same number and types of columns in the same orders. If it happens that the query you're testing returns a single column, the second argument may be an array.

**set\_has()**

SELECT set\_has( :sql, :sql, :description ); SELECT set\_has( :sql, :sql );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:description

A short description of the test.

When you need to test that a query returns at least some subset of records, set\_has() is the hammer you're looking for. It tests that the the results of a query contain at least the results returned by another query, if not more. That is, the test passes if the second query's results are a subset of the first query's results. The second query can even return an empty set, in which case the test will pass no matter what the first query returns. Not very useful perhaps, but set-theoretically correct.

As with set\_eq() . the SQL arguments can be the names of prepared statements or strings containing an SQL query (see the [summary](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Pursuing+Your+Query) for query argument details), or one of each. If it happens that the query you're testing returns a single column, the second argument may be an array.

In whatever case, a failing test will yield useful diagnostics just like:

# Failed test 122 # Missing records: # (44,Anna) # (86,Angelina)

As with set\_eq() , set\_has() will also provide useful diagnostics when the queries return incompatible columns. Internally, it uses an EXCEPT query to determine if there any any unexpectedly missing results.

**set\_hasnt()**

SELECT set\_hasnt( :sql, :sql, :description ); SELECT set\_hasnt( :sql, :sql );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:description

A short description of the test.

This test function is the inverse of set\_has() : the test passes when the results of the first query have none of the results of the second query. Diagnostics are similarly useful:

# Failed test 198 # Extra records: # (44,Anna) # (86,Angelina)

Internally, the function uses an INTERSECT query to determine if there is any unexpected overlap between the query results.

**bag\_eq()**

SELECT bag\_eq( :sql, :sql, :description ); SELECT bag\_eq( :sql, :sql ); SELECT bag\_eq( :sql, :array, :description ); SELECT bag\_eq( :sql, :array );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:array

An array of values representing a single-column row values.

:description

A short description of the test.

The bag\_eq() function is just like set\_eq() , except that it considers the results as bags rather than as sets. A bag is a set that allows duplicates. In practice, it mean that you can use bag\_eq() to test result sets where order doesn't matter, but duplication does. In other words, if a two rows are the same in the first result set, the same row must appear twice in the second result set.

Otherwise, this function behaves exactly like set\_eq() , including the utility of its diagnostics.

**bag\_ne()**

SELECT bag\_ne( :sql, :sql, :description ); SELECT bag\_ne( :sql, :sql ); SELECT bag\_ne( :sql, :array, :description ); SELECT bag\_ne( :sql, :array );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:array

An array of values representing a single-column row values.

:description

A short description of the test.

The inverse of bag\_eq() , this function tests that the results of two queries are *not* the same, including duplicates. The two queries can as usual be the names of prepared statements or strings containing an SQL query (see the [summary](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Pursuing+Your+Query) for query argument details), or even one of each. The two queries, however, must return results that are directly comparable — that is, with the same number and types of columns in the same orders. If it happens that the query you're testing returns a single column, the second argument may be an array.

**bag\_has()**

SELECT bag\_has( :sql, :sql, :description ); SELECT bag\_has( :sql, :sql );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:description

A short description of the test.

The bag\_has() function is just like set\_has() , except that it considers the results as bags rather than as sets. A bag is a set with duplicates. What practice this means that you can use bag\_has() to test result sets where order doesn't matter, but duplication does. Internally, it uses an EXCEPT ALL query to determine if there any any unexpectedly missing results.

**bag\_hasnt()**

SELECT bag\_hasnt( :sql, :sql, :description ); SELECT bag\_hasnt( :sql, :sql );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:description

A short description of the test.

This test function is the inverse of bag\_hasnt() : the test passes when the results of the first query have none of the results of the second query. Diagnostics are similarly useful:

# Failed test 198 # Extra records: # (44,Anna) # (86,Angelina)

Internally, the function uses an INTERSECT ALL query to determine if there is any unexpected overlap between the query results. This means that a duplicate row in the first query will appear twice in the diagnostics if it is also duplicated in the second query.

**is\_empty()**

SELECT is\_empty( :sql, :description ); SELECT is\_empty( :sql );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:description

A short description of the test.

The is\_empty() function takes a single query string or prepared statement name as its first argument, and tests that said query returns no records. Internally it simply executes the query and if there are any results, the test fails and the results are displayed in the failure diagnostics, like so:

# Failed test 494: "Should have no inactive users" # Records returned: # (1,Jacob,false) # (2,Emily,false)

**isnt\_empty()**

SELECT isnt\_empty( :sql, :description ); SELECT isnt\_empty( :sql );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:description

A short description of the test.

This function is the inverse of is\_empty() . The test passes if the specified query, when executed, returns at least one row. If it returns no rows, the test fails.

**row\_eq()**

SELECT row\_eq( :sql, :record, :description ); SELECT row\_eq( :sql, :record );

**Parameters**

:sql

An SQL statement or the name of a prepared statement, passed as a string.

:record

A row or value, also known as a [composite type](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.postgresql.org/docs/current/static/rowtypes.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhg_9Ceyecnsvz8fYFS4IYDGhKTafw) .

:description

A short description of the test.

Compares the contents of a single row to a record. On PostgreSQL 11 and later, a bare RECORD value may be passed:

SELECT row\_eq( $$ SELECT 1, 'foo' $$, ROW(1, 'foo') );

ue to the limitations of non-C functions in earlier versions of PostgreSQL, a bare RECORD value cannot be passed to the function. You must instead pass in a valid composite type value, and cast the record argument (the second argument) to the same type. Both explicitly created composite types and table types are supported. Thus, you can do this:

CREATE TYPE sometype AS ( id INT, name TEXT ); SELECT row\_eq( $$ SELECT 1, 'foo' $$, ROW(1, 'foo')::sometype );

And, of course, this:

CREATE TABLE users ( id INT, name TEXT ); INSERT INTO users VALUES (1, 'theory'); PREPARE get\_user AS SELECT \* FROM users LIMIT 1; SELECT row\_eq( 'get\_user', ROW(1, 'theory')::users );

Compatible types can be compared, though. So if the users table actually included an active column, for example, and you only wanted to test the id and name , you could do this:

SELECT row\_eq( $$ SELECT id, name FROM users $$, ROW(1, 'theory')::sometype );

Note the use of the sometype composite type for the second argument. The upshot is that you can create composite types in your tests explicitly for comparing the return values of your queries, if such queries don't return an existing valid type.

Hopefully someday in the future we'll be able to support arbitrary record arguments. In the meantime, this is the 90% solution.

Diagnostics on failure are similar to those from is() :

# Failed test 322 # have: (1,Jacob) # want: (1,Larry)

**The Schema Things**

Need to make sure that your database is designed just the way you think it should be? Use these test functions and rest easy.

Una nota sobre comparaciones: pgTAP utiliza una prueba de equivalencia simple ( =) para comparar todos los identificadores de SQL, como los nombres de tablas, esquemas, funciones, índices y columnas (pero no los tipos de datos). Por lo tanto, en general, siempre debe usar cadenas en minúsculas al pasar argumentos de identificador a las funciones a continuación. Use cadenas de mayúsculas y minúsculas solo cuando los objetos se declararon en su esquema con comillas dobles. Por ejemplo, si creaste una tabla así:

CREATE TABLE Foo (id integer);

Luego *debes* probarlo usando solo caracteres en minúsculas (si quieres que la prueba pase):

SELECT has\_table('foo');

Sin embargo, si declaró la tabla utilizando una cadena entre comillas dobles, así:

CREATE TABLE "Foo" (id integer);

Entonces deberías probarlo usando exactamente la misma cadena, incluido el caso, así:

SELECT has\_table('Foo');

En general, esto no debería ser un problema, ya que los objetos de casos mixtos se crean raramente. Por lo tanto, si te limitas a los argumentos en minúsculas de estas funciones, deberías estar en buena forma.

**Me opongo**

En un entorno de desarrollo ocupado, es posible que tenga varios usuarios que realizan cambios en el esquema de la base de datos. A veces tienes que trabajar realmente para mantener a estas personas en línea. Por ejemplo, ¿agregan objetos a la base de datos sin agregar pruebas? ¿Tiran objetos que no deberían? Estas afirmaciones están diseñadas para ayudarlo a garantizar que los objetos en la base de datos sean exactamente los objetos que deberían estar en la base de datos, ni más ni menos.

Each tests tests that all of the objects in the database are only the objects that *should* be there. In other words, given a list of objects, say tables in a call to tables\_are() , this assertion will fail if there are tables that are not in the list, or if there are tables in the list that are missing from the database. It can also be useful for testing replication and the success or failure of schema change deployments.

If you're more interested in the specifics of particular objects, skip to the next section.

**tablespaces\_are()**

SELECT tablespaces\_are( :tablespaces, :description ); SELECT tablespaces\_are( :tablespaces );

**Parameters**

:tablespaces

An array of tablespace names.

:description

A short description of the test.

This function tests that all of the tablespaces in the database only the tablespaces that *should* be there. Ejemplo:

SELECT tablespaces\_are(ARRAY[ 'dbspace', 'indexspace' ]);

En el caso de una falla, verá los diagnósticos que listan los espacios de tabla adicionales y / o faltantes, como por ejemplo:

# Failed test 121: "There should be the correct tablespaces" # Extra tablespaces: # trigspace # Missing tablespaces: # indexspace

**schemas\_are()**

SELECT schemas\_are( :schemas, :description ); SELECT schemas\_are( :schemas );

**Parámetros**

:schemas

Una matriz de nombres de esquema.

:description

Una breve descripción de la prueba.

Esta función prueba que todos los esquemas en la base de datos solo los esquemas que *deberían* estar allí, excluyendo los esquemas del sistema y information\_schema. Ejemplo:

SELECT schemas\_are(ARRAY[ 'public', 'contrib', 'tap' ]);

En el caso de una falla, verá los diagnósticos que listan los esquemas extra y / o faltantes, como:

# Failed test 106: "There should be the correct schemas" # Extra schemas: # \_\_howdy\_\_ # Missing schemas: # someschema

**tables\_are()**

SELECT tables\_are( :schema, :tables, :description ); SELECT tables\_are( :schema, :tables ); SELECT tables\_are( :tables, :description ); SELECT tables\_are( :tables );

**Parámetros**

:schema

Nombre de un esquema en el que buscar tablas.

:tables

Una matriz de nombres de tablas.

:description

Una breve descripción de la prueba.

Esta función prueba que todas las tablas en el esquema nombrado, o que están visibles en la ruta de búsqueda, son solo las tablas que *deberían* estar allí. Si :schemase omite el argumento, se buscarán tablas en la ruta de búsqueda, excluyendo pg\_catalogy information\_schemaSi se omite la descripción, se generará una descripción predeterminada generalmente útil. Ejemplo:

SELECT tables\_are( 'myschema', ARRAY[ 'users', 'widgets', 'gadgets', 'session' ] );

En el caso de una falla, verá los diagnósticos que listan las tablas adicionales y / o faltantes, como:

# Failed test 91: "Schema public should have the correct tables" # Extra tables: # mallots # \_\_test\_table # Missing tables: # users # widgets

**partitions\_are()**

SELECT partitions\_are( :schema, :table, :partitions :description ); SELECT partitions\_are( :schema, :table, :partitions ); SELECT partitions\_are( :table, :partitions :description ); SELECT partitions\_are( :table, :partitions );

**Parámetros**

:schema

Nombre de un esquema en el que se encuentra la tabla particionada.

:table

Nombre de una tabla particionada.

:partitions

Una matriz de nombres de tablas de particiones.

:description

Una breve descripción de la prueba.

This function tests that the named table has all of the specified partitions, and those are its only partitions. The test casts partition names to the regclass type; therefore, partition names should be specified relative to the search path. Those in the search path should not be schema-qualified, while those outside the search path should be schema-qualified. Partition names and schemas should be appropriately quoted as identifiers where appropriate.

If the :schema argument is omitted, the partitioned table must be visible the search path. If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT partitions\_are( 'myschema', 'mylog', ARRAY[ 'log1', 'log2', 'log3', 'log4' ] );

Example for partitions outside the search path and requiring identifier-quoting:

SELECT partitions\_are( 'myschema', 'MyLog', ARRAY[ 'hidden."Log 1"', 'hidden."Log 2"' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing partitions, like so:

# Failed test 12: "Table myschema should have the correct partitions" # Extra partitions: # part4 # hidden."Part 5" # Missing partitions: # part5 # part6

**foreign\_tables\_are()**

SELECT foreign\_tables\_are( :schema, :foreign\_tables, :description ); SELECT foreign\_tables\_are( :schema, :foreign\_tables ); SELECT foreign\_tables\_are( :foreign\_tables, :description ); SELECT foreign\_tables\_are( :foreign\_tables );

**Parameters**

:schema

Name of a schema in which to find foreign tables.

:foreign\_tables

An array of foreign table names.

:description

A short description of the test.

This function tests that all of the foreign tables in the named schema, or that are visible in the search path, are only the foreign tables that *should* be there. If the :schema argument is omitted, foreign tables will be sought in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT foreign\_tables\_are( 'myschema', ARRAY[ 'users', 'widgets', 'gadgets', 'session' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing foreign tables, like so:

# Failed test 91: "Schema public should have the correct foreign tables" # Extra foreign tables: # mallots # \_\_test\_table # Missing foreign tables: # users # widgets

**views\_are()**

SELECT views\_are( :schema, :views, :description ); SELECT views\_are( :schema, :views ); SELECT views\_are( :views, :description ); SELECT views\_are( :views );

**Parameters**

:schema

Name of a schema in which to find views.

:views

An array of view names.

:description

A short description of the test.

This function tests that all of the views in the named schema, or that are visible in the search path, are only the views that *should* be there. If the :schema argument is omitted, views will be sought in the search path, excluding pg\_catalog and information\_schema If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT views\_are( 'myschema', ARRAY[ 'users', 'widgets', 'gadgets', 'session' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing views, like so:

# Failed test 92: "Schema public should have the correct views" # Extra views: # v\_userlog\_tmp # \_\_test\_view # Missing views: # v\_userlog # eated

**materialized\_views\_are()**

SELECT materialized\_views\_are( :schema, :materialized\_views, :description ); SELECT materialized\_views\_are( :schema, :materialized\_views ); SELECT materialized\_views\_are( :materialized\_views, :description ); SELECT materialized\_views\_are( :materialized\_views );

**Parameters**

:schema

Name of a schema in which to find materialized views.

:materialized\_views

An array of materialized view names.

:description

A short description of the test.

This function tests that all of the materialized views in the named schema, or that are visible in the search path, are only the materialized views that *should* be there. If the :schema argument is omitted, materialized views will be sought in the search path, excluding pg\_catalog and information\_schema If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT materialized\_views\_are( 'myschema', ARRAY[ 'users', 'widgets', 'gadgets', 'session' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing materialized views, like so:

# Failed test 92: "Schema public should have the correct materialized views" # Extra materialized views: # v\_userlog\_tmp # \_\_test\_view # Missing materialized views: # v\_userlog # eated

**sequences\_are()**

SELECT sequences\_are( :schema, :sequences, :description ); SELECT sequences\_are( :schema, :sequences ); SELECT sequences\_are( :sequences, :description ); SELECT sequences\_are( :sequences );

**Parameters**

:schema

Name of a schema in which to find sequences.

:sequences

An array of sequence names.

:description

A short description of the test.

Esta función prueba que todas las secuencias en el esquema nombrado, o que son visibles en la ruta de búsqueda, son solo las secuencias que *deberían* estar allí. Si :schemase omite el argumento, se buscarán secuencias en la ruta de búsqueda, excluyendo pg\_catalogy information\_schema. Si se omite la descripción, se generará una descripción predeterminada generalmente útil. Ejemplo:

SELECT sequences\_are( 'myschema', ARRAY[ 'users', 'widgets', 'gadgets', 'session' ] );

En el caso de una falla, verá diagnósticos que listan las secuencias adicionales y / o faltantes, como:

# Failed test 93: "Schema public should have the correct sequences" # These are extra sequences: # seq\_mallots # \_\_test\_table\_seq # These sequences are missing: # users\_seq # widgets\_seq

**columns\_are()**

SELECT columns\_are( :schema, :table, :columns, :description ); SELECT columns\_are( :schema, :table, :columns ); SELECT columns\_are( :table, :columns, :description ); SELECT columns\_are( :table, :columns );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :table.

:table

Nombre de una tabla en la que buscar columnas.

:columns

Una matriz de nombres de columna.

:description

Una breve descripción de la prueba.

Esta función prueba que todas las columnas en la tabla nombrada son solo las columnas que *deben* estar en esa tabla. Si :schemase omite el argumento, la tabla debe estar visible en la ruta de búsqueda, excluyendo pg\_catalogy information\_schema. Si se omite la descripción, se generará una descripción predeterminada generalmente útil. Ejemplo:

SELECT columns\_are( 'myschema', 'atable', ARRAY[ 'id', 'name', 'rank', 'sn' ] );

En el caso de una falla, verá los diagnósticos que listan las columnas adicionales y / o faltantes, como así:

# Failed test 183: "Table users should have the correct columns" # Extra columns: # given\_name # surname # Missing columns: # name

**indexes\_are()**

SELECT indexes\_are( :schema, :table, :indexes, :description ); SELECT indexes\_are( :schema, :table, :indexes ); SELECT indexes\_are( :table, :indexes, :description ); SELECT indexes\_are( :table, :indexes );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :table.

:table

Nombre de una tabla en la que encontrar los índices.

:indexes

Una matriz de nombres de índice.

:description

Una breve descripción de la prueba.

This function tests that all of the indexes on the named table are only the indexes that *should* be on that table. If the :schema argument is omitted, the table must be visible in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT indexes\_are( 'myschema', 'atable', ARRAY[ 'atable\_pkey', 'idx\_atable\_name' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing indexes, like so:

# Failed test 180: "Table fou should have the correct indexes" # Extra indexes: # fou\_pkey # Missing indexes: # idx\_fou\_name

**triggers\_are()**

SELECT triggers\_are( :schema, :table, :triggers, :description ); SELECT triggers\_are( :schema, :table, :triggers ); SELECT triggers\_are( :table, :triggers, :description ); SELECT triggers\_are( :table, :triggers );

**Parameters**

:schema

Name of a schema in which to find the :table .

:table

Name of a table in which to find triggers.

:triggers

An array of trigger names.

:description

A short description of the test.

This function tests that all of the triggers on the named table are only the triggers that *should* be on that table. If the :schema argument is omitted, the table must be visible in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT triggers\_are( 'myschema', 'atable', ARRAY[ 'atable\_pkey', 'idx\_atable\_name' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing triggers, like so:

# Failed test 180: "Table fou should have the correct triggers" # Extra triggers: # set\_user\_pass # Missing triggers: # set\_users\_pass

**functions\_are()**

SELECT functions\_are( :schema, :functions, :description ); SELECT functions\_are( :schema, :functions ); SELECT functions\_are( :functions, :description ); SELECT functions\_are( :functions );

**Parameters**

:schema

Name of a schema in which to find functions.

:functions

An array of function names.

:description

A short description of the test.

This function tests that all of the functions in the named schema, or that are visible in the search path, are only the functions that *should* be there. If the :schema argument is omitted, functions will be sought in the search path, excluding pg\_catalog and information\_schema If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT functions\_are( 'myschema', ARRAY[ 'foo', 'bar', 'frobnitz' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing functions, like so:

# Failed test 150: "Schema someschema should have the correct functions" # Extra functions: # schnauzify # Missing functions: # frobnitz

**roles\_are()**

SELECT roles\_are( :roles, :description ); SELECT roles\_are( :roles );

**Parameters**

:roles

An array of role names.

:description

A short description of the test.

This function tests that all of the roles in the database only the roles that *should* be there. Ejemplo:

SELECT roles\_are(ARRAY[ 'postgres', 'someone', 'root' ]);

In the event of a failure, you'll see diagnostics listing the extra and/or missing roles, like so:

# Failed test 195: "There should be the correct roles" # Extra roles: # root # Missing roles: # bobby

**users\_are()**

SELECT users\_are( :users, :description ); SELECT users\_are( :users );

**Parameters**

:users

An array of user names.

:description

A short description of the test.

This function tests that all of the users in the database only the users that *should* be there. Ejemplo:

SELECT users\_are(ARRAY[ 'postgres', 'someone', 'root' ]);

In the event of a failure, you'll see diagnostics listing the extra and/or missing users, like so:

# Failed test 195: "There should be the correct users" # Extra users: # root # Missing users: # bobby

**groups\_are()**

SELECT groups\_are( :groups, :description ); SELECT groups\_are( :groups );

**Parameters**

:groups

An array of group names.

:description

A short description of the test.

This function tests that all of the groups in the database only the groups that *should* be there. Ejemplo:

SELECT groups\_are(ARRAY[ 'postgres', 'admins, 'l0s3rs' ]);

In the event of a failure, you'll see diagnostics listing the extra and/or missing groups, like so:

# Failed test 210: "There should be the correct groups" # Extra groups: # meanies # Missing groups: # \_\_howdy\_\_

**languages\_are()**

SELECT languages\_are( :languages, :description ); SELECT languages\_are( :languages );

**Parameters**

:languages

An array of language names.

:description

A short description of the test.

This function tests that all of the languages in the database only the languages that *should* be there. Ejemplo:

SELECT languages\_are(ARRAY[ 'plpgsql', 'plperl', 'pllolcode' ]);

In the event of a failure, you'll see diagnostics listing the extra and/or missing languages, like so:

# Failed test 225: "There should be the correct procedural languages" # Extra languages: # pllolcode # Missing languages: # plpgsql

**opclasses\_are()**

SELECT opclasses\_are( :schema, :opclasses, :description ); SELECT opclasses\_are( :schema, :opclasses ); SELECT opclasses\_are( :opclasses, :description ); SELECT opclasses\_are( :opclasses );

**Parameters**

:schema

Name of a schema in which to find opclasses.

:opclasses

An array of opclass names.

:description

A short description of the test.

This function tests that all of the operator classes in the named schema, or that are visible in the search path, are only the opclasses that *should* be there. If the :schema argument is omitted, opclasses will be sought in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT opclasses\_are( 'myschema', ARRAY[ 'foo', 'bar', 'frobnitz' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing opclasses, like so:

# Failed test 251: "Schema public should have the correct operator classes" # Extra operator classes: # goofy\_ops # Missing operator classes: # custom\_ops

**rules\_are()**

SELECT rules\_are( :schema, :table, :rules, :description ); SELECT rules\_are( :schema, :table, :rules ); SELECT rules\_are( :table, :rules, :description ); SELECT rules\_are( :table, :rules );

**Parameters**

:schema

Name of a schema in which to find the :table .

:table

Name of a table in which to find rules.

:rules

An array of rule names.

:description

A short description of the test.

This function tests that all of the rules on the named relation are only the rules that *should* be on that relation (a table, view or a materialized view). If the :schema argument is omitted, the rules must be visible in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT rules\_are( 'myschema', 'atable', ARRAY[ 'on\_insert', 'on\_update', 'on\_delete' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing rules, like so:

# Failed test 281: "Relation public.users should have the correct rules" # Extra rules: # on\_select # Missing rules: # on\_delete

**types\_are()**

SELECT types\_are( :schema, :types, :description ); SELECT types\_are( :schema, :types ); SELECT types\_are( :types, :description ); SELECT types\_are( :types );

**Parameters**

:schema

Name of a schema in which to find types.

:types

An array of data type names.

:description

A short description of the test.

Tests that all of the types in the named schema are the only types in that schema, including base types, composite types, domains, enums, and pseudo-types. If the :schema argument is omitted, the types must be visible in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT types\_are('myschema', ARRAY[ 'timezone', 'state' ]);

In the event of a failure, you'll see diagnostics listing the extra and/or missing types, like so:

# Failed test 307: "Schema someschema should have the correct types" # Extra types: # sometype # Missing types: # timezone

**domains\_are()**

SELECT domains\_are( :schema, :domains, :description ); SELECT domains\_are( :schema, :domains ); SELECT domains\_are( :domains, :description ); SELECT domains\_are( :domains );

**Parameters**

:schema

Name of a schema in which to find domains.

:domains

An array of data domain names.

:description

A short description of the test.

Tests that all of the domains in the named schema are the only domains in that schema. If the :schema argument is omitted, the domains must be visible in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT domains\_are('myschema', ARRAY[ 'timezone', 'state' ]);

In the event of a failure, you'll see diagnostics listing the extra and/or missing domains, like so:

# Failed test 327: "Schema someschema should have the correct domains" # Extra domains: # somedomain # Missing domains: # timezone

**enums\_are()**

SELECT enums\_are( :schema, :enums, :description ); SELECT enums\_are( :schema, :enums ); SELECT enums\_are( :enums, :description ); SELECT enums\_are( :enums );

**Parameters**

:schema

Name of a schema in which to find enums.

:enums

An array of enum data type names.

:description

A short description of the test.

Tests that all of the enums in the named schema are the only enums in that schema. Enums are supported in PostgreSQL 8.3 and up. If the :schema argument is omitted, the enums must be visible in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT enums\_are('myschema', ARRAY[ 'timezone', 'state' ]);

In the event of a failure, you'll see diagnostics listing the extra and/or missing enums, like so:

# Failed test 333: "Schema someschema should have the correct enums" # Extra enums: # someenum # Missing enums: # bug\_status

**casts\_are()**

SELECT casts\_are( :casts, :description ); SELECT casts\_are( :casts );

**Parameters**

:casts

An array of cast names.

:description

A short description of the test.

Esta función prueba que todos los lanzamientos en la base de datos son solo los lanzamientos que *deberían* estar en esa base de datos. Los cast son especificados como cadenas en una sintaxis similar a como son declarados vía CREATE CAST. El patrón es :source\_type AS :target\_type. Si cualquiera de los tipos se creó con comillas dobles para forzar mayúsculas y minúsculas o caracteres especiales, debe usar comillas dobles en las cadenas de conversión. Ejemplo:

SELECT casts\_are(ARRAY[ 'integer AS "myInteger"', 'integer AS double precision', 'integer AS reltime', 'integer AS numeric', ]);

Si se omite la descripción, se generará una descripción predeterminada generalmente útil.

En el caso de una falla, verá diagnósticos que listan los modelos extra y / o faltantes, así:

# Failed test 302: "There should be the correct casts" # Extra casts: # lseg AS point # Missing casts: # lseg AS integer

**operators\_are()**

SELECT operators\_are( :schema, :operators, :description ); SELECT operators\_are( :schema, :operators ); SELECT operators\_are( :operators, :description ); SELECT operators\_are( :operators );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar operadores.

:operators

Una matriz de operadores.

:description

Una breve descripción de la prueba.

Tests that all of the operators in the named schema are the only operators in that schema. If the :schema argument is omitted, the operators must be visible in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated.

The :operators argument is specified as an array of strings in which each operator is defined similarly to the display of the :regoperator type. The format is :op(:leftop,:rightop) RETURNS :return\_type .

For left operators the left argument type should be NONE . For right operators, the right argument type should be NONE . The example above shows one one of each of the operator types. =(citext,citext) is an infix operator, -(bigint,NONE) is a left operator, and !(NONE,bigint) is a right operator. Ejemplo:

SELECT operators\_are( 'public', ARRAY[ '=(citext,citext) RETURNS boolean', '-(NONE,bigint) RETURNS bigint', '!(bigint,NONE) RETURNS numeric' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing operators, like so:

# Failed test 453: "Schema public should have the correct operators" # Extra operators: # +(integer,integer) RETURNS integer # Missing enums: # +(integer,text) RETURNS text

**extensions\_are()**

SELECT extensions\_are( :schema, :extensions, :description ); SELECT extensions\_are( :schema, :extensions ); SELECT extensions\_are( :extensions, :description ); SELECT extensions\_are( :extensions );

**Parameters**

:schema

Name of a schema associated with the extensions.

:extensions

An array of extension names.

:description

A short description of the test.

This function tests all of the extensions that should be present. If :schema is specified, it will test only for extensions associated the named schema (via the schema parameter in the extension's control file, ov the WITH SCHEMA clause of the [CREATE EXTENSION](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://www.postgresql.org/docs/current/static/extend-extensions.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhhPCoc4SdQeluATF4Vl1gbjCd2oqw) statement). Otherwise it will check for all extension in the database, including pgTAP itself. If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT extensions\_are( 'myschema', ARRAY[ 'citext', 'isn', 'plpgsql' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing extensions, like so:

# Failed test 91: "Schema public should have the correct extensions" # Extra extensions: # pgtap # ltree # Missing extensions: # citext # isn

**To Have or Have Not**

Perhaps you're not so concerned with ensuring the [precise correlation of database objects](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#I+Object!) . Perhaps you just need to make sure that certain objects exist (or that certain objects *don't* exist). Has venido al lugar correcto.

**has\_tablespace()**

SELECT has\_tablespace( :tablespace, :location, :description ); SELECT has\_tablespace( :tablespace, :description ); SELECT has\_tablespace( :tablespace );

**Parameters**

:tablespace

Name of a tablespace.

:location

The tablespace's Location on disk.

:description

A short description of the test.

This function tests whether or not a tablespace exists in the database. The first argument is a tablespace name. The second is either the a file system path for the database or a test description. If you specify a location path, you must pass a description as the third argument; otherwise, if you omit the test description, it will be set to “Tablespace :tablespace should exist”. Ejemplo:

SELECT has\_tablespace('sometablespace', '/data/dbs');

**hasnt\_tablespace()**

SELECT hasnt\_tablespace( :tablespace, :description ); SELECT hasnt\_tablespace( :tablespace );

**Parameters**

:tablespace

Name of a tablespace.

:description

A short description of the test.

This function is the inverse of has\_tablespace() . The test passes if the specified tablespace does *not* exist.

**has\_schema()**

SELECT has\_schema( :schema, :description ); SELECT has\_schema( :schema );

**Parameters**

:schema

Name of a schema.

:description

A short description of the test.

This function tests whether or not a schema exists in the database. The first argument is a schema name and the second is the test description. If you omit the test description, it will be set to “Schema :schema should exist”.

**hasnt\_schema()**

SELECT hasnt\_schema( 'someschema', 'There should be no schema someschema' );

**Parameters**

:schema

Name of a schema.

:description

A short description of the test.

This function is the inverse of has\_schema() . The test passes if the specified schema does *not* exist.

**has\_relation()**

SELECT has\_relation( :schema, :relation, :description ); SELECT has\_relation( :relation, :description ); SELECT has\_relation( :relation );

**Parameters**

:schema

Name of a schema in which to find the relation.

:relation

Name of a relation.

:description

A short description of the test.

Esta función prueba si existe o no una relación en la base de datos. Las relaciones son tablas, vistas, vistas materializadas, secuencias, tipos compuestos, tablas externas y tablas de brindis. El primer argumento es un nombre de esquema, el segundo es un nombre de relación y el tercero es la descripción de la prueba. Si omite el esquema, la relación debe estar visible en la ruta de búsqueda. Ejemplo:

SELECT has\_relation('myschema', 'somerelation');

Si omite la descripción de la prueba, se establecerá en "La relación :relationdebe existir".

**hasnt\_relation()**

SELECT hasnt\_relation( :schema, :relation, :description ); SELECT hasnt\_relation( :relation, :description ); SELECT hasnt\_relation( :relation );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar la relación.

:relation

Nombre de una relación.

:description

Una breve descripción de la prueba.

Esta función es la inversa de has\_relation(). La prueba pasa si la relación especificada *no* existe.

**has\_table()**

SELECT has\_table( :schema, :table, :description ); SELECT has\_table( :schema, :table ); SELECT has\_table( :table, :description ); SELECT has\_table( :table );

**Parámetros**

:schema

Nombre de un esquema en el que se encuentra la tabla.

:table

Nombre de una tabla.

:description

Una breve descripción de la prueba.

Esta función prueba si existe o no una tabla en la base de datos. El primer argumento es un nombre de esquema, el segundo es un nombre de tabla y el tercero es la descripción de la prueba. Si omite el esquema, la tabla debe estar visible en la ruta de búsqueda. Ejemplo:

SELECT has\_table('myschema'::name, 'sometable'::name);

Si omite la descripción de la prueba, se establecerá en "La tabla :tabledebería existir".

Tenga en cuenta que esta función no reconocerá tablas ajenas; Se usa has\_foreign\_table()para probar la presencia de tablas extranjeras.

**hasnt\_table()**

SELECT hasnt\_table( :schema, :table, :description ); SELECT hasnt\_table( :schema, :table ); SELECT hasnt\_table( :table, :description ); SELECT hasnt\_table( :table );

**Parámetros**

:schema

Nombre de un esquema en el que se encuentra la tabla.

:table

Nombre de una tabla.

:description

Una breve descripción de la prueba.

Esta función es la inversa de has\_table(). La prueba pasa si la tabla especificada *no* existe.

**has\_view()**

SELECT has\_view( :schema, :view, :description ); SELECT has\_view( :view, :description ); SELECT has\_view( :view );

**Parámetros**

:schema

Nombre de un esquema en el que buscar la vista.

:view

Nombre de una vista.

:description

Una breve descripción de la prueba.

This function tests whether or not a view exists in the database. The first argument is a schema name, the second is a view name, and the third is the test description. If you omit the schema, the view must be visible in the search path. Ejemplo:

SELECT has\_view('myschema', 'someview');

If you omit the test description, it will be set to “View :view should exist”.

**hasnt\_view()**

SELECT hasnt\_view( :schema, :view, :description ); SELECT hasnt\_view( :view, :description ); SELECT hasnt\_view( :view );

**Parameters**

:schema

Name of a schema in which to find the view.

:view

Name of a view.

:description

A short description of the test.

This function is the inverse of has\_view() . The test passes if the specified view does *not* exist.

**has\_materialized\_view()**

SELECT has\_materialized\_view( :schema, :materialized\_view, :description ); SELECT has\_materialized\_view( :materialized\_view, :description ); SELECT has\_materialized\_view( :materialized\_view );

**Parameters**

:schema

Name of a schema in which to find the materialized view.

:materialized\_view

Name of a materialized view.

:description

A short description of the test.

This function tests whether or not a materialized view exists in the database. The first argument is a schema name, the second is a materialized view name, and the third is the test description. If you omit the schema, the materialized view must be visible in the search path. Ejemplo:

SELECT has\_materialized\_view('myschema', 'some\_materialized\_view');

If you omit the test description, it will be set to “Materialized view :materialized\_view should exist”.

**hasnt\_materialized\_view()**

SELECT hasnt\_materialized\_view( :schema, :materialized\_view, :description ); SELECT hasnt\_materialized\_view( :materialized\_view, :description ); SELECT hasnt\_materialized\_view( :materialized\_view );

**Parameters**

:schema

Name of a schema in which to find the materialized view.

:materialized\_view

Name of a materialized view.

:description

A short description of the test.

This function is the inverse of has\_view() . The test passes if the specified materialized view does *not* exist.

**has\_inherited\_tables()**

SELECT has\_inherited\_tables( :schema, :table, :description ); SELECT has\_inherited\_tables( :schema, :table ); SELECT has\_inherited\_tables( :table, :description ); SELECT has\_inherited\_tables( :table );

**Parameters**

:schema

Name of a schema in which to search for the table that has children.

:table

Name of the table that must have children.

:description

A description of the test.

This function checks that the specified table has other tables that inherit from it. If you find that the function call confuses the table name for a description, cast the table to the NAME type:

SELECT has\_inherited\_tables('myschema', 'sometable'::NAME);

**hasnt\_inherited\_tables()**

SELECT hasnt\_inherited\_tables( :schema, :table, :description ); SELECT hasnt\_inherited\_tables( :schema, :table ); SELECT hasnt\_inherited\_tables( :table, :description ); SELECT hasnt\_inherited\_tables( :table );

**Parameters**

:schema

Name of a schema in which to search for the table that must not have children.

:table

Name of the table that must not have children.

:description

A description of the test.

This function checks that the specified table has no tables inheriting from it. It is the opposite of the function has\_inherited\_tables() . If you find that the function call confuses the table name for a description, cast the table to the NAME type:

SELECT hasnt\_inherited\_tables('myschema', 'sometable'::NAME);

**is\_ancestor\_of()**

SELECT is\_ancestor\_of( :ancestor\_schema, :ancestor\_table, :descendent\_schema, :descendent\_table, :depth, :description ); SELECT is\_ancestor\_of( :ancestor\_schema, :ancestor\_table, :descendent\_schema, :descendent\_table, :depth ); SELECT is\_ancestor\_of( :ancestor\_schema, :ancestor\_table, :descendent\_schema, :descendent\_table, :description ); SELECT is\_ancestor\_of( :ancestor\_schema, :ancestor\_table, :descendent\_schema, :descendent\_table ); SELECT is\_ancestor\_of( :ancestor\_table, :descendent\_table, :depth, :description ); SELECT is\_ancestor\_of( :ancestor\_table, :descendent\_table, :depth ); SELECT is\_ancestor\_of( :ancestor\_table, :descendent\_table, :description ); SELECT is\_ancestor\_of( :ancestor\_table, :descendent\_table );

**Parameters**

:ancestor\_schema

Name of the schema in which the ancestor table must be found.

:ancestor\_table

Name of the ancestor table.

:descendent\_schema

Name of the schema in which the descendent table must be found.

:descendent\_table

Name of the descendent table.

:depth

La distancia de herencia entre las dos tablas.

:description

Descripción de la prueba.

Esta función comprueba si la tabla marcada como "antecesora" es efectivamente una tabla de la que se hereda la tabla "descendiente", que existe una cadena de herencia entre las dos tablas. El argumento de profundidad opcional especifica la longitud de la cadena de herencia entre las tablas; Si no se especifica, la distancia de herencia puede ser de cualquier longitud. Si :descriptionse omite, se creará un sustituto razonable.

Si encuentra que la llamada a la función parece estar confundiéndose, convierta la secuencia al NAMEtipo:

SELECT is\_ancestor\_of('myschema', 'ancestor', 'myschema', 'descendent'::NAME);

**isnt\_ancestor\_of()**

SELECT isnt\_ancestor\_of( :ancestor\_schema, :ancestor\_table, :descendent\_schema, :descendent\_table, :depth, :description ); SELECT isnt\_ancestor\_of( :ancestor\_schema, :ancestor\_table, :descendent\_schema, :descendent\_table, :depth ); SELECT isnt\_ancestor\_of( :ancestor\_schema, :ancestor\_table, :descendent\_schema, :descendent\_table, :description ); SELECT isnt\_ancestor\_of( :ancestor\_schema, :ancestor\_table, :descendent\_schema, :descendent\_table ); SELECT isnt\_ancestor\_of( :ancestor\_table, :descendent\_table, :depth, :description ); SELECT isnt\_ancestor\_of( :ancestor\_table, :descendent\_table, :depth ); SELECT isnt\_ancestor\_of( :ancestor\_table, :descendent\_table, :description ); SELECT isnt\_ancestor\_of( :ancestor\_table, :descendent\_table );

**Parámetros**

:ancestor\_schema

Nombre del esquema en el que se debe encontrar la tabla de ancestro.

:ancestor\_table

Nombre de la tabla de antepasados.

:descendent\_schema

Nombre del esquema en el que se debe encontrar la tabla descendente.

:descendent\_table

Nombre de la tabla descendiente.

:depth

La distancia de herencia entre las dos tablas.

:description

Descripción de la prueba.

Esta función garantiza que la tabla marcada como "antecesor" no es una tabla de la que "descendiente" hereda, que no hay una cadena de herencia entre las dos tablas. Si se pasa el argumento de profundidad opcional, la prueba solo asegura que las dos tablas no están relacionadas a esa distancia; Todavía podrían ser una relación de herencia entre ellos. Si :descriptionse omite, se creará un sustituto razonable.

Si encuentra que la llamada a la función parece estar confundiéndose, convierta la secuencia al NAMEtipo:

SELECT isnt\_ancestor\_of('myschema', 'ancestor', 'myschema', 'descendent'::NAME);

**is\_descendent\_of()**

SELECT is\_descendent\_of( :descendent\_schema, :descendent\_table, :ancestor\_schema, :ancestor\_table, :depth, :description ); SELECT is\_descendent\_of( :descendent\_schema, :descendent\_table, :ancestor\_schema, :ancestor\_table, :depth ); SELECT is\_descendent\_of( :descendent\_schema, :descendent\_table, :ancestor\_schema, :ancestor\_table, :description ); SELECT is\_descendent\_of( :descendent\_schema, :descendent\_table, :ancestor\_schema, :ancestor\_table ); SELECT is\_descendent\_of( :descendent\_table, :ancestor\_table, :depth, :description ); SELECT is\_descendent\_of( :descendent\_table, :ancestor\_table, :depth ); SELECT is\_descendent\_of( :descendent\_table, :ancestor\_table, :description ); SELECT is\_descendent\_of( :descendent\_table, :ancestor\_table );

**Parámetros**

:descendent\_schema

Nombre del esquema en el que se debe encontrar la tabla descendente.

:descendent\_table

Nombre de la tabla descendiente.

:ancestor\_schema

Nombre del esquema en el que se debe encontrar la tabla de ancestro.

:ancestor\_table

Nombre de la tabla de antepasados.

:depth

La distancia de herencia entre las dos tablas.

:description

Descripción de la prueba.

This function provide exactly the same functionality as is\_ancestor\_of() , but with the ancestor and descendent arguments swapped.

**isnt\_descendent\_of()**

SELECT isnt\_descendent\_of( :descendent\_schema, :descendent\_table, :ancestor\_schema, :ancestor\_table, :depth, :description ); SELECT isnt\_descendent\_of( :descendent\_schema, :descendent\_table, :ancestor\_schema, :ancestor\_table, :depth ); SELECT isnt\_descendent\_of( :descendent\_schema, :descendent\_table, :ancestor\_schema, :ancestor\_table, :description ); SELECT isnt\_descendent\_of( :descendent\_table, :ancestor\_table, :depth, :description ); SELECT isnt\_descendent\_of( :descendent\_table, :ancestor\_table, :depth ); SELECT isnt\_descendent\_of( :descendent\_table, :ancestor\_table, :description ); SELECT isnt\_descendent\_of( :descendent\_table, :ancestor\_table );

**Parameters**

:ancestor\_schema

Name of the schema in which the ancestor table must be found.

:ancestor\_table

Name of the ancestor table.

:descendent\_schema

Name of the schema in which the descendent table must be found.

:descendent\_table

Name of the descendent table.

:depth

The inheritance distance between the two tables.

:description

Description of the test.

This function provide exactly the same functionality as isnt\_ancestor\_of() , but with the ancestor and descendent arguments swapped.

**has\_sequence()**

SELECT has\_sequence( :schema, :sequence, :description ); SELECT has\_sequence( :schema, :sequence ); SELECT has\_sequence( :sequence, :description ); SELECT has\_sequence( :sequence );

**Parameters**

:schema

Name of a schema in which to find the sequence.

:sequence

Name of a sequence.

:description

A short description of the test.

This function tests whether or not a sequence exists in the database. The first argument is a schema name, the second is a sequence name, and the third is the test description. If you omit the schema, the sequence must be visible in the search path. Ejemplo:

SELECT has\_sequence('somesequence');

If you omit the test description, it will be set to “Sequence :schema . :sequence should exist”. If you find that the function call seems to be getting confused, cast the sequence to the NAME type:

SELECT has\_sequence('myschema', 'somesequence'::NAME);

**hasnt\_sequence()**

SELECT hasnt\_sequence( :schema, :sequence, :description ); SELECT hasnt\_sequence( :sequence, :description ); SELECT hasnt\_sequence( :sequence );

**Parameters**

:schema

Name of a schema in which to find the sequence.

:sequence

Name of a sequence.

:description

A short description of the test.

This function is the inverse of has\_sequence() . The test passes if the specified sequence does *not* exist.

**has\_foreign\_table()**

SELECT has\_foreign\_table( :schema, :table, :description ); SELECT has\_foreign\_table( :schema, :table ); SELECT has\_foreign\_table( :table, :description ); SELECT has\_foreign\_table( :table );

**Parameters**

:schema

Name of a schema in which to find the foreign table.

:table

Name of a foreign table.

:description

A short description of the test.

This function tests whether or not a foreign table exists in the database. The first argument is a schema name, the second is a foreign table name, and the third is the test description. If you omit the schema, the foreign table must be visible in the search path. Ejemplo:

SELECT has\_foreign\_table('myschema'::name, 'some\_foreign\_table'::name);

If you omit the test description, it will be set to “Foreign table :table should exist”.

**hasnt\_foreign\_table()**

SELECT hasnt\_foreign\_table( :schema, :table, :description ); SELECT hasnt\_foreign\_table( :schema, :table ); SELECT hasnt\_foreign\_table( :table, :description ); SELECT hasnt\_foreign\_table( :table );

**Parameters**

:schema

Name of a schema in which to find the foreign table.

:table

Name of a foreign table.

:description

A short description of the test.

This function is the inverse of has\_foreign\_table() . The test passes if the specified foreign table does *not* exist.

**has\_type()**

SELECT has\_type( schema, type, description ); SELECT has\_type( schema, type ); SELECT has\_type( type, description ); SELECT has\_type( type );

**Parameters**

:schema

Name of a schema in which to find the data type.

:type

Name of a data type.

:description

A short description of the test.

This function tests whether or not a type exists in the database. Detects all types of types, including base types, composite types, domains, enums, and pseudo-types. The first argument is a schema name, the second is a type name, and the third is the test description. If you omit the schema, the type must be visible in the search path. If you omit the test description, it will be set to “Type :type should exist”. If you're passing a schema and type rather than type and description, be sure to cast the arguments to name values so that your type name doesn't get treated as a description. Ejemplo:

SELECT has\_type( 'myschema', 'sometype' );

If you've created a composite type and want to test that the composed types are a part of it, use the column testing functions to verify them, like so:

CREATE TYPE foo AS (id int, name text); SELECT has\_type( 'foo' ); SELECT has\_column( 'foo', 'id' ); SELECT col\_type\_is( 'foo', 'id', 'integer' );

**hasnt\_type()**

SELECT hasnt\_type( schema, type, description ); SELECT hasnt\_type( schema, type ); SELECT hasnt\_type( type, description ); SELECT hasnt\_type( type );

**Parameters**

:schema

Name of a schema in which to find the data type.

:type

Name of a data type.

:description

A short description of the test.

This function is the inverse of has\_type() . The test passes if the specified type does *not* exist.

**has\_composite()**

SELECT has\_composite( schema, type, description ); SELECT has\_composite( schema, type ); SELECT has\_composite( type, description ); SELECT has\_composite( type );

**Parameters**

:schema

Name of a schema in which to find the composite type.

:composite type

Name of a composite type.

:description

A short description of the test.

This function tests whether or not a composite type exists in the database. The first argument is a schema name, the second is the name of a composite type, and the third is the test description. If you omit the schema, the composite type must be visible in the search path. If you omit the test description, it will be set to “Composite type :composite type should exist”. Ejemplo:

SELECT has\_composite( 'myschema', 'somecomposite' );

Si está pasando un esquema y un tipo compuesto en lugar de un tipo compuesto y una descripción, asegúrese de convertir los argumentos en namevalores para que su nombre de tipo compuesto no sea tratado como una descripción.

**hasnt\_composite()**

SELECT hasnt\_composite( schema, type, description ); SELECT hasnt\_composite( schema, type ); SELECT hasnt\_composite( type, description ); SELECT hasnt\_composite( type );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el tipo compuesto.

:composite type

Nombre de un tipo compuesto.

:description

Una breve descripción de la prueba.

Esta función es la inversa de has\_composite(). La prueba pasa si el tipo compuesto especificado *no* existe.

**has\_domain()**

SELECT has\_domain( schema, domain, description ); SELECT has\_domain( schema, domain ); SELECT has\_domain( domain, description ); SELECT has\_domain( domain );

**Parámetros**

:schema

Nombre de un esquema en el que se encuentra el dominio.

:domain

Nombre de un dominio.

:description

Una breve descripción de la prueba.

Esta función prueba si un dominio existe o no en la base de datos. El primer argumento es un nombre de esquema, el segundo es el nombre de un dominio y el tercero es la descripción de la prueba. Si omite el esquema, el dominio debe estar visible en la ruta de búsqueda. Si omite la descripción de la prueba, se establecerá en "El dominio :domaindebe existir". Ejemplo:

SELECT has\_domain( 'myschema', 'somedomain' );

Si está pasando un esquema y un dominio en lugar de un dominio y una descripción, asegúrese de convertir los argumentos en namevalores para que su nombre de dominio no sea tratado como una descripción.

**hasnt\_domain()**

SELECT hasnt\_domain( schema, domain, description ); SELECT hasnt\_domain( schema, domain ); SELECT hasnt\_domain( domain, description ); SELECT hasnt\_domain( domain );

**Parámetros**

:schema

Nombre de un esquema en el que se encuentra el dominio.

:domain

Nombre de un dominio.

:description

Una breve descripción de la prueba.

Esta función es la inversa de has\_domain(). La prueba pasa si el dominio especificado *no* existe.

**has\_enum()**

SELECT has\_enum( schema, enum, description ); SELECT has\_enum( schema, enum ); SELECT has\_enum( enum, description ); SELECT has\_enum( enum );

**Parámetros**

:schema

Nombre de un esquema en el que se encuentra la enumeración.

:enum

Name of a enum.

:description

A short description of the test.

This function tests whether or not a enum exists in the database. Enums are supported in PostgreSQL 8.3 or higher. The first argument is a schema name, the second is the an enum name, and the third is the test description. If you omit the schema, the enum must be visible in the search path. If you omit the test description, it will be set to “Enum :enum should exist”. Ejemplo:

SELECT has\_enum( 'myschema', 'someenum' );

If you're passing a schema and enum rather than enum and description, be sure to cast the arguments to name values so that your enum name doesn't get treated as a description.

**hasnt\_enum()**

SELECT hasnt\_enum( schema, enum, description ); SELECT hasnt\_enum( schema, enum ); SELECT hasnt\_enum( enum, description ); SELECT hasnt\_enum( enum );

**Parameters**

:schema

Name of a schema in which to find the enum.

:enum

Name of a enum.

:description

A short description of the test.

This function is the inverse of has\_enum() . The test passes if the specified enum does *not* exist.

**has\_index()**

SELECT has\_index( :schema, :table, :index, :columns, :description ); SELECT has\_index( :schema, :table, :index, :columns ); SELECT has\_index( :schema, :table, :index, :column, :description ); SELECT has\_index( :schema, :table, :index, :column ); SELECT has\_index( :schema, :table, :index, :description ); SELECT has\_index( :schema, :table, :index ); SELECT has\_index( :table, :index, :columns, :description ); SELECT has\_index( :table, :index, :columns ); SELECT has\_index( :table, :index, :column, :description ); SELECT has\_index( :table, :index, :column ); SELECT has\_index( :table, :index, :description ); SELECT has\_index( :table, :index );

**Parameters**

:schema

Nombre de un esquema en el que encontrar el índice.

:table

Nombre de una tabla en la que se encuentra el índice.

:index

Nombre de un índice.

:columns

Array de las columnas y / o expresiones en el índice.

:column

Nombre o expresión de columna indexada.

:description

Una breve descripción de la prueba.

Comprueba la existencia de un índice con nombre asociado con la tabla con nombre. El :schemaargumento es opcional, al igual que el nombre de la columna o nombres o expresiones, y la descripción. El argumento de las columnas puede ser una cadena que nombra una columna o expresión, o una matriz de nombres y / o expresiones de columna. Para las expresiones, debe usar minúsculas para que todas las palabras clave y funciones de SQL se comparen correctamente con la forma interna de la expresión de PostgreSQL. Las expresiones no funcionales también se deben envolver entre paréntesis. Algunos ejemplos:

SELECT has\_index( 'myschema', 'sometable', 'myindex', ARRAY[ 'somecolumn', 'anothercolumn', 'lower(txtcolumn)' ], 'Index "myindex" should exist' ); SELECT has\_index('myschema', 'sometable', 'anidx', 'somecolumn'); SELECT has\_index('myschema', 'sometable', 'loweridx', '(somearray[1])'); SELECT has\_index('sometable', 'someindex');

Si encuentra que la llamada a la función parece estar confundiéndose, convierta el nombre del índice al NAMEtipo:

SELECT has\_index( 'public', 'sometab', 'idx\_foo', 'name'::name );

Si el índice no existe, se has\_index()mostrará un mensaje de diagnóstico como:

# Index "blah" ON public.sometab not found

Si se encontró el índice pero la especificación o expresión de la columna es incorrecta, los diagnósticos se verán más así:

# have: "idx\_baz" ON public.sometab(lower(name)) # want: "idx\_baz" ON public.sometab(lower(lname))

**hasnt\_index()**

SELECT hasnt\_index( schema, table, index, description ); SELECT hasnt\_index( schema, table, index ); SELECT hasnt\_index( table, index, description ); SELECT hasnt\_index( table, index );

**Parámetros**

:schema

Nombre de un esquema en el que no se encuentra el índice.

:table

Nombre de una tabla en la que no se encuentra el índice.

:index

Nombre de un índice.

:description

Una breve descripción de la prueba.

Esta función es la inversa de has\_index(). La prueba pasa si el índice especificado *no* existe.

**has\_trigger()**

SELECT has\_trigger( :schema, :table, :trigger, :description ); SELECT has\_trigger( :schema, :table, :trigger ); SELECT has\_trigger( :table, :trigger, :description ); SELECT has\_trigger( :table, :trigger )` ###

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el disparador.

:table

Nombre de una tabla en la que se encuentra el disparador.

:trigger

Nombre de un disparador.

:description

Una breve descripción de la prueba.

Prueba para ver si la tabla especificada tiene el disparador nombrado. El :descriptiones opcional, y si se omite el esquema, la mesa con el que está asociado el disparador debe ser visible en la ruta de búsqueda.

**hasnt\_trigger()**

SELECT hasnt\_trigger( :schema, :table, :trigger, :description ); SELECT hasnt\_trigger( :schema, :table, :trigger ); SELECT hasnt\_trigger( :table, :trigger, :description ); SELECT hasnt\_trigger( :table, :trigger )` ###

**Parámetros**

:schema

Nombre de un esquema en el que no se encuentra el activador.

:table

Nombre de una tabla en la que no se encuentra el disparador.

:trigger

Nombre de un disparador.

:description

Una breve descripción de la prueba.

Esta función es la inversa de has\_trigger(). La prueba pasa si el disparador especificado *no* existe.

**has\_rule()**

SELECT has\_rule( :schema, :table, :rule, :description ); SELECT has\_rule( :schema, :table, :rule ); SELECT has\_rule( :table, :rule, :description ); SELECT has\_rule( :table, :rule )` ###

**Parámetros**

:schema

Nombre de un esquema en el que se encuentra la regla.

:table

Nombre de una tabla en la que se encuentra la regla.

:rule

Nombre de una regla.

:description

Una breve descripción de la prueba.

Pruebas para ver si la tabla especificada tiene la regla nombrada. El :descriptiones opcional, y si se omite el esquema, la mesa con la que está asociada la regla debe ser visible en la ruta de búsqueda.

**hasnt\_rule()**

SELECT hasnt\_rule( :schema, :table, :rule, :description ); SELECT hasnt\_rule( :schema, :table, :rule ); SELECT hasnt\_rule( :table, :rule, :description ); SELECT hasnt\_rule( :table, :rule )` ###

**Parámetros**

:schema

Name of a schema in which to not find the rule.

:table

Name of a table in which to not find the rule.

:rule

Name of an rule.

:description

A short description of the test.

This function is the inverse of has\_rule() . The test passes if the specified rule does *not* exist.

**has\_function()**

SELECT has\_function( :schema, :function, :args, :description ); SELECT has\_function( :schema, :function, :args ); SELECT has\_function( :schema, :function, :description ); SELECT has\_function( :schema, :function ); SELECT has\_function( :function, :args, :description ); SELECT has\_function( :function, :args ); SELECT has\_function( :function, :description ); SELECT has\_function( :function );

**Parameters**

:schema

Name of a schema in which to find the function.

:function

Name of a function.

:args

Array of data types of the function arguments.

:description

A short description of the test.

Checks to be sure that the given function exists in the named schema and with the specified argument data types. If :schema is omitted, has\_function() will search for the function in the schemas defined in the search path. If :args is omitted, has\_function() will see if the function exists without regard to its arguments. Some examples:

SELECT has\_function( 'pg\_catalog', 'decode', ARRAY[ 'text', 'text' ], 'Function decode(text, text) should exist' ); SELECT has\_function( 'do\_something' ); SELECT has\_function( 'do\_something', ARRAY['integer'] ); SELECT has\_function( 'do\_something', ARRAY['numeric'] );

The :args argument should be formatted as it would be displayed in the view of a function using the \df command in psql . For example, even if you have a numeric column with a precision of 8, you should specify ARRAY['numeric'] . If you created a varchar(64) column, you should pass the :args argument as ARRAY['character varying'] .

If you wish to use the two-argument form of has\_function() , specifying only the schema and the function name, you must cast the :function argument to :name in order to disambiguate it from from the has\_function(:function, :description) form. If you neglect to do so, your results will be unexpected.

Also, if you use the string form to specify the :args array, be sure to cast it to name to disambiguate it from a text string:

SELECT has\_function( 'lower', '{text}'::name[] );

**Deprecation notice:** The old name for this test function, can\_ok() , is still available, but emits a warning when called. It will be removed in a future version of pgTAP.

**hasnt\_function()**

SELECT hasnt\_function( :schema, :function, :args, :description ); SELECT hasnt\_function( :schema, :function, :args ); SELECT hasnt\_function( :schema, :function, :description ); SELECT hasnt\_function( :schema, :function ); SELECT hasnt\_function( :function, :args, :description ); SELECT hasnt\_function( :function, :args ); SELECT hasnt\_function( :function, :description ); SELECT hasnt\_function( :function );

**Parameters**

:schema

Name of a schema in which not to find the function.

:function

Name of a function.

:args

Array of data types of the function arguments.

:description

A short description of the test.

This function is the inverse of has\_function() . The test passes if the specified function (optionally with the specified signature) does *not* exist.

**has\_cast()**

SELECT has\_cast( :source\_type, :target\_type, :schema, :function, :description ); SELECT has\_cast( :source\_type, :target\_type, :schema, :function ); SELECT has\_cast( :source\_type, :target\_type, :function, :description ); SELECT has\_cast( :source\_type, :target\_type, :function ); SELECT has\_cast( :source\_type, :target\_type, :description ); SELECT has\_cast( :source\_type, :target\_type );

**Parameters**

:source\_type

Data type of the source value without typemod.

:target\_type

Data type of the target value without typemod.

:schema

Schema in which to find the operator function.

:function

Name of the operator function.

:description

A short description of the test.

Tests for the existence of a cast. A cast consists of a source data type, a target data type, and perhaps a (possibly schema-qualified) function. An example:

SELECT has\_cast( 'integer', 'bigint', 'pg\_catalog', 'int8' );

If you omit the description for the 3- or 4-argument version, you'll need to cast the function name to the NAME data type so that PostgreSQL doesn't resolve the function name as a description. Por ejemplo:

SELECT has\_cast( 'integer', 'bigint', 'int8'::NAME );

pgTAP will generate a useful description if you don't provide one.

Note that pgTAP does not compare typemods. So if you wanted to test for a cast between, say, a uuid type and bit(128) , this will not work:

SELECT has\_cast( 'integer', 'bit(128)' );

But this will:

SELECT has\_cast( 'integer', 'bit' );

**hasnt\_cast()**

SELECT hasnt\_cast( :source\_type, :target\_type, :schema, :function, :description ); SELECT hasnt\_cast( :source\_type, :target\_type, :schema, :function ); SELECT hasnt\_cast( :source\_type, :target\_type, :function, :description ); SELECT hasnt\_cast( :source\_type, :target\_type, :function ); SELECT hasnt\_cast( :source\_type, :target\_type, :description ); SELECT hasnt\_cast( :source\_type, :target\_type );

**Parameters**

:source\_type

Data type of the source value.

:target\_type

Data type of the target value.

:schema

Schema in which not to find the operator function.

:function

Name of the operator function.

:description

A short description of the test.

This function is the inverse of has\_cast() : the test passes if the specified cast does *not* exist.

**has\_operator()**

SELECT has\_operator( :left\_type, :schema, :name, :right\_type, :return\_type, :description ); SELECT has\_operator( :left\_type, :schema, :name, :right\_type, :return\_type ); SELECT has\_operator( :left\_type, :name, :right\_type, :return\_type, :description ); SELECT has\_operator( :left\_type, :name, :right\_type, :return\_type ); SELECT has\_operator( :left\_type, :name, :right\_type, :description ); SELECT has\_operator( :left\_type, :name, :right\_type );

**Parameters**

:left\_type

Data type of the left operand.

:schema

Schema in which to find the operator.

:name

Name of the operator.

:right\_type

Data type of the right operand.

:return\_type

Data type of the return value.

:description

A short description of the test.

Tests for the presence of a binary operator. If the operator exists with the given schema, name, left and right arguments, and return value, the test will fail. If the operator does not exist, the test will fail. Ejemplo:

SELECT has\_operator( 'integer', 'pg\_catalog', '<=', 'integer', 'boolean' );

If you omit the schema name, then the operator must be visible in the search path. If you omit the test description, pgTAP will generate a reasonable one for you. The return value is also optional. If you need to test for a left or right unary operator, use has\_leftop() or has\_rightop() instead.

**has\_leftop()**

SELECT has\_leftop( :schema, :name, :type, :return\_type, :description ); SELECT has\_leftop( :schema, :name, :type, :return\_type ); SELECT has\_leftop( :name, :type, :return\_type, :description ); SELECT has\_leftop( :name, :type, :return\_type ); SELECT has\_leftop( :name, :type, :description ); SELECT has\_leftop( :name, :type );

**Parameters**

:schema

Schema in which to find the operator.

:name

Name of the operator.

:type

Data type of the operand.

:return\_type

Data type of the return value.

:description

A short description of the test.

Tests for the presence of a left-unary operator. If the operator exists with the given schema, name, right argument, and return value, the test will fail. If the operator does not exist, the test will fail. Ejemplo:

SELECT has\_leftop( 'pg\_catalog', '!!', 'bigint', 'numeric' );

If you omit the schema name, then the operator must be visible in the search path. If you omit the test description, pgTAP will generate a reasonable one for you. The return type is also optional.

**has\_rightop()**

SELECT has\_rightop( :schema, :name, :type, :return\_type, :description ); SELECT has\_rightop( :schema, :name, :type, :return\_type ); SELECT has\_rightop( :name, :type, :return\_type, :description ); SELECT has\_rightop( :name, :type, :return\_type ); SELECT has\_rightop( :name, :type, :description ); SELECT has\_rightop( :name, :type );

**Parameters**

:schema

Schema in which to find the operator.

:name

Name of the operator.

:type

Data type of the operand.

:return\_type

Data type of the return value.

:description

A short description of the test.

Tests for the presence of a right-unary operator. If the operator exists with the given left argument, schema, name, and return value, the test will fail. If the operator does not exist, the test will fail. Ejemplo:

SELECT has\_rightop( 'bigint', 'pg\_catalog', '!', 'numeric' );

If you omit the schema name, then the operator must be visible in the search path. If you omit the test description, pgTAP will generate a reasonable one for you. The return type is also optional.

**has\_opclass()**

SELECT has\_opclass( :schema, :name, :description ); SELECT has\_opclass( :schema, :name ); SELECT has\_opclass( :name, :description ); SELECT has\_opclass( :name );

**Parameters**

:schema

Schema in which to find the operator class.

:name

Name of the operator class.

:description

A short description of the test.

Tests for the presence of an operator class. If you omit the schema name, then the operator must be visible in the search path. If you omit the test description, pgTAP will generate a reasonable one for you. The return value is also optional.

**hasnt\_opclass()**

SELECT hasnt\_opclass( :schema, :name, :description ); SELECT hasnt\_opclass( :schema, :name ); SELECT hasnt\_opclass( :name, :description ); SELECT hasnt\_opclass( :name );

**Parameters**

:schema

Schema in which not to find the operator class.

:name

Name of the operator class.

:description

A short description of the test.

This function is the inverse of has\_opclass() . The test passes if the specified operator class does *not* exist.

**has\_role()**

SELECT has\_role( :role, :description ); SELECT has\_role( :role );

**Parameters**

:role

Name of the role.

:description

A short description of the test.

Checks to ensure that a database role exists. If the description is omitted, it will default to “Role :role should exist”.

**hasnt\_role()**

SELECT hasnt\_role( :role, :description ); SELECT hasnt\_role( :role );

**Parameters**

:role

Name of the role.

:description

A short description of the test.

The inverse of has\_role() , this function tests for the *absence* of a database role.

**has\_user()**

SELECT has\_user( :user, :description ); SELECT has\_user( :user );

**Parameters**

:user

Name of the user.

:description

A short description of the test.

Comprueba para asegurarse de que existe un usuario de base de datos. Si se omite la descripción, se establecerá de forma predeterminada en "El usuario :userdebe existir".

**hasnt\_user()**

SELECT hasnt\_user( :user, :description ); SELECT hasnt\_user( :user );

**Parámetros**

:user

Nombre del usuario.

:description

Una breve descripción de la prueba.

A la inversa de has\_user(), esta función prueba la *ausencia* de un usuario de la base de datos.

**has\_group()**

SELECT has\_group( :group, :description ); SELECT has\_group( :group );

**Parámetros**

:group

Nombre del grupo.

:description

Una breve descripción de la prueba.

Comprueba para asegurarse de que existe un grupo de base de datos. Si se omite la descripción, se establecerá de manera predeterminada "Grupo :groupdebería existir".

**hasnt\_group()**

SELECT hasnt\_group( :group, :description ); SELECT hasnt\_group( :group );

**Parámetros**

:group

Nombre del grupo.

:description

Una breve descripción de la prueba.

A la inversa de has\_group(), esta función prueba la *ausencia* de un grupo de base de datos.

**has\_language()**

SELECT has\_language( :language, :description ); SELECT has\_language( :language );

**Parámetros**

:language

Nombre de la lengua.

:description

Una breve descripción de la prueba.

Comprueba para asegurarse de que existe un lenguaje procesal. Si se omite la descripción, se establecerá de manera predeterminada en "El lenguaje de procedimiento :languagedebería existir".

**hasnt\_language()**

SELECT hasnt\_language( :language, :description ); SELECT hasnt\_language( :language );

**Parámetros**

:language

Nombre de la lengua.

:description

Una breve descripción de la prueba.

A la inversa de has\_language(), esta función prueba la *ausencia* de un lenguaje de procedimiento.

**has\_extension()**

SELECT has\_extension( :schema, :extension, :description ); SELECT has\_extension( :schema, :extension ); SELECT has\_extension( :extension, :description ); SELECT has\_extension( :extension );

**Parámetros**

:schema

Esquema en el que se instalaron los objetos de la extensión.

:extension

Nombre de una extensión.

:description

Una breve descripción de la prueba.

Esta función prueba si existe o no una extensión en la base de datos. El primer argumento es el esquema en el que se instalaron los objetos de extensión, el segundo el nombre de la extensión y el tercero la descripción de la prueba. Si se omite el esquema, puede estar asociado con cualquier esquema o sin esquema. Si se omite la descripción de la prueba, se establecerá en "La extensión :extensiondebería existir". Ejemplo:

SELECT has\_extension('public', 'pgtap');

**hasnt\_extension()**

SELECT hasnt\_extension( :schema, :extension, :description ); SELECT hasnt\_extension( :schema, :extension ); SELECT hasnt\_extension( :extension, :description ); SELECT hasnt\_extension( :extension );

**Parámetros**

:schema

Esquema en el que se instalarían los objetos de la extensión.

:extension

Nombre de una extensión.

:description

Una breve descripción de la prueba.

Esta función es la inversa de has\_extension(). La prueba pasa si la extensión especificada *no* existe.

**Table For One**

Okay, you're sure that your database has exactly the [right schema](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#I+Object!) and that all of the objects you need [are there](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#To+Have+or+Have+Not) . So let's take a closer look at tables. There are a lot of ways to look at tables, to make sure that they have all the columns, indexes, constraints, keys, and indexes they need. So we have the assertions to validate 'em.

**has\_column()**

SELECT has\_column( :schema, :table, :column, :description ); SELECT has\_column( :table, :column, :description ); SELECT has\_column( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:column

Name of the column.

:description

A short description of the test.

Tests whether or not a column exists in a given table, view, materialized view or composite type. The first argument is the schema name, the second the table name, the third the column name, and the fourth is the test description. If the schema is omitted, the table must be visible in the search path. If the test description is omitted, it will be set to “Column :table.:column should exist”.

**hasnt\_column()**

SELECT hasnt\_column( :schema, :table, :column, :description ); SELECT hasnt\_column( :table, :column, :description ); SELECT hasnt\_column( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:column

Name of the column.

:description

A short description of the test.

This function is the inverse of has\_column() . The test passes if the specified column does *not* exist in the specified table, view, materialized view or composite type.

**col\_not\_null()**

SELECT col\_not\_null( :schema, :table, :column, :description ); SELECT col\_not\_null( :table, :column, :description ); SELECT col\_not\_null( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:column

Name of the column.

:description

A short description of the test.

Tests whether the specified column has a NOT NULL constraint. The first argument is the schema name, the second the table name, the third the column name, and the fourth is the test description. If the schema is omitted, the table must be visible in the search path. If the test description is omitted, it will be set to “Column :table.:column should be NOT NULL”. Note that this test will fail with a useful diagnostic message if the table or column in question does not exist. But use has\_column() to make sure the column exists first, eh?

**col\_is\_null()**

SELECT col\_is\_null( :schema, :table, :column, :description ); SELECT col\_is\_null( :table, :column, :description ); SELECT col\_is\_null( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:column

Name of the column.

:description

A short description of the test.

This function is the inverse of col\_not\_null() : the test passes if the column does not have a NOT NULL constraint. The first argument is the schema name, the second the table name, the third the column name, and the fourth is the test description. If the schema is omitted, the table must be visible in the search path. If the test description is omitted, it will be set to “Column :table.:column should allow NULL”. Note that this test will fail with a useful diagnostic message if the table or column in question does not exist. But use has\_column() to make sure the column exists first, eh?

**col\_has\_default()**

SELECT col\_has\_default( :schema, :table, :column, :description ); SELECT col\_has\_default( :table, :column, :description ); SELECT col\_has\_default( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:column

Name of the column.

:description

A short description of the test.

Tests whether or not a column has a default value. Fails if the column doesn't have a default value. It will also fail if the column doesn't exist, and emit useful diagnostics to let you know:

# Failed test 136: "desc" # Column public.sometab.\_\_asdfasdfs\_\_ does not exist

**col\_hasnt\_default()**

SELECT col\_hasnt\_default( :schema, :table, :column, :description ); SELECT col\_hasnt\_default( :table, :column, :description ); SELECT col\_hasnt\_default( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:column

Name of the column.

:description

A short description of the test.

This function is the inverse of col\_has\_default() . The test passes if the specified column does *not* have a default. It will still fail if the column does not exist, and emit useful diagnostics to let you know.

**col\_type\_is()**

SELECT col\_type\_is( :schema, :table, :column, :type\_schema, :type, :description ); SELECT col\_type\_is( :schema, :table, :column, :type\_schema, :type ); SELECT col\_type\_is( :schema, :table, :column, :type, :description ); SELECT col\_type\_is( :schema, :table, :column, :type ); SELECT col\_type\_is( :table, :column, :type, :description ); SELECT col\_type\_is( :table, :column, :type );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:column

Name of the column.

:type\_schema

Schema in which to find the data type.

:type

Name of a data type.

:description

A short description of the test.

This function tests that the specified column is of a particular type. If it fails, it will emit diagnostics naming the actual type. The first argument is the schema name, the second the table name, the third the column name, the fourth the type's schema, the fifth the type, and the sixth is the test description.

If the table schema is omitted, the table must be visible in the search path. If the type schema is omitted, it must be visible in the search path; otherwise, the diagnostics will report the schema it's actually in. The schema can optionally be included in the :type argument, eg, 'contrib.citext .

If the test description is omitted, it will be set to “Column :schema.:table.:column should be type :schema.:type ”. Note that this test will fail if the table or column in question does not exist.

The type argument should be formatted as it would be displayed in the view of a table using the \d command in psql . For example, if you have a numeric column with a precision of 8, you should specify “numeric(8,0)”. If you created a varchar(64) column, you should pass the type as “character varying(64)”. Ejemplo:

SELECT col\_type\_is( 'myschema', 'sometable', 'somecolumn', 'numeric(10,2)' );

If the test fails, it will output useful diagnostics. For example this test:

SELECT col\_type\_is( 'pg\_catalog', 'pg\_type', 'typname', 'text' );

Will produce something like this:

# Failed test 138: "Column pg\_catalog.pg\_type.typname should be type text" # have: name # want: text

It will even tell you if the test fails because a column doesn't exist or actually has no default. But use has\_column() to make sure the column exists first, eh?

**col\_default\_is()**

SELECT col\_default\_is( :schema, :table, :column, :default, :description ); SELECT col\_default\_is( :table, :column, :default, :description ); SELECT col\_default\_is( :table, :column, :default );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:column

Name of the column.

:default

Default value expressed as a string.

:description

A short description of the test.

Tests the default value of a column. If it fails, it will emit diagnostics showing the actual default value. The first argument is the schema name, the second the table name, the third the column name, the fourth the default value, and the fifth is the test description. If the schema is omitted, the table must be visible in the search path. If the test description is omitted, it will be set to “Column :table.:column should default to :default ”. Note that this test will fail if the table or column in question does not exist.

El argumento predeterminado debe tener un tipo no ambiguo para que la llamada tenga éxito. Si ve un error como 'ERROR: no se pudo determinar el tipo polimórfico porque la entrada tiene el tipo "desconocido"', se debe a que se olvidó de emitir el valor esperado, probablemente un NULL(que, por cierto, solo puede probar adecuadamente en PostgreSQL 8.3 y posteriores), a su tipo adecuado. IOW, esto va a fallar:

SELECT col\_default\_is( 'tab', age, NULL );

Pero esto no lo hará:

SELECT col\_default\_is( 'tab', age, NULL::integer );

También puede probar los valores predeterminados funcionales. Simplemente especifique la llamada a la función como una cadena:

SELECT col\_default\_is( 'user', 'created\_at', 'now()' );

Pero tenga en cuenta que la representación de las funciones especiales de sintaxis de SQL cambió en PostgreSQL 10. Donde antes, los valores predeterminados CURRENT\_USERy los amigos solían representarse como funciones:

SELECT col\_default\_is( 'widgets', 'created\_by', '"current\_user"()' );

A partir de PostgreSQL 10, cumplen con la especificación SQL para aparecer en mayúsculas y sin paréntesis de ruta:

SELECT col\_default\_is( 'widgets', 'created\_by', 'CURRENT\_USER' );

Si necesita admitir ambas variantes, use pg\_version\_num()para decidir cuál usar:

SELECT col\_default\_is( 'widgets', 'created\_by', CASE WHEN pg\_version\_num() >= 100000 THEN 'CURRENT\_USER ELSE '"current\_user"()' END );

Consulte la nota en la documentación de [Funciones de información](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://www.postgresql.org/docs/current/functions-info.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhhIgZTSX_uEinpJe1zW9wlCTvUKCQ) del [sistema](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://www.postgresql.org/docs/current/functions-info.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhhIgZTSX_uEinpJe1zW9wlCTvUKCQ) para obtener una lista completa.

Si la prueba falla, se emitirán diagnósticos útiles. Por ejemplo, esta prueba:

SELECT col\_default\_is( 'pg\_catalog', 'pg\_type', 'typname', 'foo', 'check typname' );

Producirá algo como esto:

# Failed test 152: "check typname" # have: NULL # want: foo

Y si la prueba falla porque la tabla o columna en cuestión no existe, el diagnóstico también le dirá eso. Pero usas has\_column()y col\_has\_default()para probar esas condiciones antes de llamar col\_default\_is(), ¿verdad? *¿¿¿Derecha???* Sí, bien, eso creía yo.

**has\_pk()**

SELECT has\_pk( :schema, :table, :description ); SELECT has\_pk( :table, :description ); SELECT has\_pk( :table );

**Parámetros**

:schema

Esquema en el que encontrar la tabla.

:table

Nombre de una tabla.

:description

Una breve descripción de la prueba.

Tests whether or not a table has a primary key. The first argument is the schema name, the second the table name, the the third is the test description. If the schema is omitted, the table must be visible in the search path. If the test description is omitted, it will be set to “Table :table should have a primary key”. Note that this test will fail if the table in question does not exist.

**hasnt\_pk()**

SELECT hasnt\_pk( :schema, :table, :description ); SELECT hasnt\_pk( :table, :description ); SELECT hasnt\_pk( :table );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:description

A short description of the test.

This function is the inverse of has\_pk() . The test passes if the specified primary key does *not* exist.

**has\_fk()**

SELECT has\_fk( :schema, :table, :description ); SELECT has\_fk( :table, :description ); SELECT has\_fk( :table );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:description

A short description of the test.

Tests whether or not a table has a foreign key constraint. The first argument is the schema name, the second the table name, the the third is the test description. If the schema is omitted, the table must be visible in the search path. If the test description is omitted, it will be set to “Table :table should have a foreign key constraint”. Note that this test will fail if the table in question does not exist.

**hasnt\_fk()**

SELECT hasnt\_fk( :schema, :table, :description ); SELECT hasnt\_fk( :table, :description ); SELECT hasnt\_fk( :table );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table.

:description

A short description of the test.

This function is the inverse of has\_fk() . The test passes if the specified foreign key does *not* exist.

**col\_is\_pk()**

SELECT col\_is\_pk( :schema, :table, :columns, :description ); SELECT col\_is\_pk( :schema, :table, :column, :description ); SELECT col\_is\_pk( :table, :columns, :description ); SELECT col\_is\_pk( :table, :column, :description ); SELECT col\_is\_pk( :table, :columns ); SELECT col\_is\_pk( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table containing the primary key.

:columns

Array of the names of the primary key columns.

:column

Name of the primary key column.

:description

A short description of the test.

Tests whether the specified column or columns in a table is/are the primary key for that table. If it fails, it will emit diagnostics showing the actual primary key columns, if any. The first argument is the schema name, the second the table name, the third the column name or an array of column names, and the fourth is the test description. Ejemplos:

SELECT col\_is\_pk( 'myschema', 'sometable', 'id' ); SELECT col\_is\_pk( 'persons', ARRAY['given\_name', 'surname'] );

If the schema is omitted, the table must be visible in the search path. If the test description is omitted, it will be set to “Column :table(:column) should be a primary key”. Note that this test will fail if the table or column in question does not exist.

If the test fails, it will output useful diagnostics. For example this test:

SELECT col\_is\_pk( 'pg\_type', 'id' );

Will produce something like this:

# Failed test 178: "Column pg\_type.id should be a primary key" # have: {} # want: {id}

**col\_isnt\_pk()**

SELECT col\_isnt\_pk( :schema, :table, :columns, :description ); SELECT col\_isnt\_pk( :schema, :table, :column, :description ); SELECT col\_isnt\_pk( :table, :columns, :description ); SELECT col\_isnt\_pk( :table, :column, :description ); SELECT col\_isnt\_pk( :table, :columns ); SELECT col\_isnt\_pk( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table not containing the primary key.

:columns

Array de los nombres de las columnas de clave primaria.

:column

Nombre de la columna de clave primaria.

:description

Una breve descripción de la prueba.

Esta función es la inversa de col\_is\_pk(). La prueba pasa si la columna o columnas especificadas no son una clave principal.

**col\_is\_fk()**

SELECT col\_is\_fk( :schema, :table, :columns, :description ); SELECT col\_is\_fk( :schema, :table, :column, :description ); SELECT col\_is\_fk( :table, :columns, :description ); SELECT col\_is\_fk( :table, :column, :description ); SELECT col\_is\_fk( :table, :columns ); SELECT col\_is\_fk( :table, :column );

**Parámetros**

:schema

Esquema en el que encontrar la tabla.

:table

Nombre de una tabla que contiene la restricción de clave externa.

:columns

Array de los nombres de las columnas de clave externa.

:column

Nombre de la columna de clave externa.

:description

Una breve descripción de la prueba.

Al igual que col\_is\_fk(), excepto que prueba que la columna o matriz de columnas son una clave principal. Los diagnósticos de falla también son un poco diferentes. Como la tabla puede tener más de una clave externa, los diagnósticos simplemente listan todas las columnas de restricción de clave externa, de esta manera:

# Table widget has foreign key constraints on these columns: # {thingy\_id} # {surname,given\_name}

**col\_isnt\_fk()**

SELECT col\_isnt\_fk( :schema, :table, :columns, :description ); SELECT col\_isnt\_fk( :schema, :table, :column, :description ); SELECT col\_isnt\_fk( :table, :columns, :description ); SELECT col\_isnt\_fk( :table, :column, :description ); SELECT col\_isnt\_fk( :table, :columns ); SELECT col\_isnt\_fk( :table, :column );

**Parámetros**

:schema

Esquema en el que encontrar la tabla.

:table

Nombre de una tabla que no contiene la restricción de clave externa.

:columns

Array de los nombres de las columnas de clave externa.

:column

Nombre de la columna de clave externa.

:description

Una breve descripción de la prueba.

Esta función es la inversa de col\_is\_fk(). La prueba pasa si la columna o columnas especificadas no son una clave externa.

**fk\_ok()**

SELECT fk\_ok( :fk\_schema, :fk\_table, :fk\_columns, :pk\_schema, :pk\_table, :pk\_columns, :description ); SELECT fk\_ok( :fk\_schema, :fk\_table, :fk\_columns, :pk\_schema, :pk\_table, :pk\_columns ); SELECT fk\_ok( :fk\_table, :fk\_columns, :pk\_table, :pk\_columns, :description ); SELECT fk\_ok( :fk\_table, :fk\_columns, :pk\_table, :pk\_columns ); SELECT fk\_ok( :fk\_schema, :fk\_table, :fk\_column, :pk\_schema, :pk\_table, :pk\_column, :description ); SELECT fk\_ok( :fk\_schema, :fk\_table, :fk\_column, :pk\_schema, :pk\_table, :pk\_column ); SELECT fk\_ok( :fk\_table, :fk\_column, :pk\_table, :pk\_column, :description ); SELECT fk\_ok( :fk\_table, :fk\_column, :pk\_table, :pk\_column );

**Parámetros**

:fk\_schema

Esquema en el que encontrar la tabla con la clave externa.

:fk\_table

Nombre de una tabla que contiene la clave externa.

:fk\_columns

Array de los nombres de las columnas de clave externa.

:fk\_column

Nombre de la columna de clave externa.

:pk\_schema

Esquema en el que encontrar la tabla con la clave primaria

:pk\_table

Nombre de una tabla que contiene la clave primaria.

:pk\_columns

Array de los nombres de las columnas de clave primaria.

:pk\_column

Nombre de la columna de clave primaria.

:description

Una breve descripción de la prueba.

Esta función combina col\_is\_fk()y col\_is\_pk()en una sola prueba que también determina que existe una relación de clave externa entre las tablas de clave principal y externa. Para probar adecuadamente sus relaciones, esta debe ser su principal función de prueba de elección.

Los primeros tres argumentos son el esquema, la tabla y la columna o matriz de columnas que constituyen la restricción de clave externa. El nombre del esquema es opcional, y las columnas se pueden especificar como una cadena para una sola columna o una matriz de cadenas para varias columnas. Los siguientes tres argumentos son el esquema, la tabla y la columna o columnas que constituyen la clave primaria correspondiente. Nuevamente, el esquema es opcional y las columnas pueden ser una cadena o matriz de cadenas (aunque, por supuesto, debe tener el mismo número de elementos que el argumento de la columna de clave externa). El séptimo argumento es una descripción opcional. Si no se incluye, se establecerá como :fk\_schema.:fk\_table(:fk\_column)referencia :pk\_column.pk\_table(:pk\_column). Algunos ejemplos:

SELECT fk\_ok( 'myschema', 'sometable', 'big\_id', 'myschema', 'bigtable', 'id' ); SELECT fk\_ok( 'contacts', ARRAY['person\_given\_name', 'person\_surname'], 'persons', ARRAY['given\_name', 'surname'], );

Si la prueba falla, se emitirán diagnósticos útiles. Por ejemplo esta prueba:

SELECT fk\_ok( 'contacts', 'person\_id', 'persons', 'id' );

Producirá algo como esto:

# Failed test 178: "Column contacts(person\_id) should reference persons(id)" # have: contacts(person\_id) REFERENCES persons(id)" # want: contacts(person\_nick) REFERENCES persons(nick)"

**has\_unique()**

SELECT has\_unique( :schema, :table, :description ); SELECT has\_unique( :table, :description ); SELECT has\_unique( :table );

**Parámetros**

:schema

Schema in which to find the table.

:table

Name of a table containing the unique constraint.

:description

A short description of the test.

Tests whether or not a table has a unique constraint. The first argument is the schema name, the second the table name, the the third is the test description. If the schema is omitted, the table must be visible in the search path. If the test description is omitted, it will be set to “Table :table should have a unique constraint”. Note that this test will fail if the table in question does not exist.

**col\_is\_unique()**

SELECT col\_is\_unique( schema, table, columns, description ); SELECT col\_is\_unique( schema, table, column, description ); SELECT col\_is\_unique( schema, table, columns ); SELECT col\_is\_unique( schema, table, column ); SELECT col\_is\_unique( table, columns, description ); SELECT col\_is\_unique( table, column, description ); SELECT col\_is\_unique( table, columns ); SELECT col\_is\_unique( table, column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table containing the unique constraint.

:columns

Array of the names of the unique columns.

:column

Name of the unique column.

:description

A short description of the test.

Just like col\_is\_pk() , except that it test that the column or array of columns have a unique constraint on them. Ejemplos:

SELECT col\_is\_unique( 'contacts', ARRAY['given\_name', 'surname'] ); SELECT col\_is\_unique( 'myschema', 'sometable', 'other\_id', 'myschema.sometable.other\_id should be unique' );

If you omit the description for the 3-argument version, you'll need to cast the table and column parameters to the NAME data type so that PostgreSQL doesn't resolve the function name as a description. Por ejemplo:

SELECT col\_is\_unique( 'myschema', 'sometable'::name, 'other\_id'::name );

In the event of failure, the diagnostics will list the unique constraints that were actually found, if any:

Failed test 40: "users.email should be unique" have: {username} {first\_name,last\_name} want: {email}

**has\_check()**

SELECT has\_check( :schema, :table, :description ); SELECT has\_check( :table, :description ); SELECT has\_check( :table );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table containing the check constraint.

:description

A short description of the test.

Tests whether or not a table has a check constraint. The first argument is the schema name, the second the table name, the the third is the test description. If the schema is omitted, the table must be visible in the search path. If the test description is omitted, it will be set to “Table :table should have a check constraint”. Note that this test will fail if the table in question does not exist.

In the event of failure, the diagnostics will list the columns on the table that do have check constraints, if any:

Failed test 41: "users.email should have a check constraint" have: {username} want: {email}

**col\_has\_check()**

SELECT col\_has\_check( :schema, :table, :columns, :description ); SELECT col\_has\_check( :schema, :table, :column, :description ); SELECT col\_has\_check( :table, :columns, :description ); SELECT col\_has\_check( :table, :column, :description ); SELECT col\_has\_check( :table, :columns ); SELECT col\_has\_check( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table containing the check constraint.

:columns

Array of the names of the check constraint columns.

:column

Name of the check constraint column.

:description

A short description of the test.

Al igual que col\_is\_pk(), excepto que comprueba que la columna o la matriz de columnas tienen una restricción de verificación.

**index\_is\_unique()**

SELECT index\_is\_unique( :schema, :table, :index, :description ); SELECT index\_is\_unique( :schema, :table, :index ); SELECT index\_is\_unique( :table, :index ); SELECT index\_is\_unique( :index );

**Parámetros**

:schema

Esquema en el que encontrar la tabla.

:table

Nombre de una tabla que contiene el índice.

:index

Nombre del índice.

:description

Una breve descripción de la prueba.

Comprueba si un índice es único.

**index\_is\_primary()**

SELECT index\_is\_primary( :schema, :table, :index, :description ); SELECT index\_is\_primary( :schema, :table, :index ); SELECT index\_is\_primary( :table, :index ); SELECT index\_is\_primary( :index );

**Parámetros**

:schema

Esquema en el que encontrar la tabla.

:table

Nombre de una tabla que contiene el índice.

:index

Nombre del índice.

:description

Una breve descripción de la prueba.

Comprueba si un índice está en una clave primaria.

**is\_partitioned()**

SELECT is\_partitioned( :schema, :table, :description ); SELECT is\_partitioned( :schema, :table ); SELECT is\_partitioned( :table, :description); SELECT is\_partitioned( :table );

**Parámetros**

:schema

Esquema en el que encontrar la tabla particionada.

:table

Nombre de una tabla particionada.

:description

Una breve descripción de la prueba.

Comprueba si una tabla está particionada. El primer argumento es el nombre del esquema, el segundo el nombre de la tabla, el tercero es la descripción de la prueba. Si se omite el esquema, la tabla debe estar visible en la ruta de búsqueda. Si se omite la descripción de la prueba, se establecerá en "La tabla :tabledebe particionarse". Tenga en cuenta que esta prueba fallará si la tabla en cuestión no existe.

**isnt\_partitioned()**

SELECT isnt\_partitioned( :schema, :table, :description ); SELECT isnt\_partitioned( :schema, :table ); SELECT isnt\_partitioned( :table, :description); SELECT isnt\_partitioned( :table );

**Parámetros**

:schema

Esquema en el que encontrar la tabla.

:table

Nombre de una tabla.

:description

Una breve descripción de la prueba.

Esta función es la inversa de is\_partitioned(). La prueba pasa si la tabla especificada *no* está particionada, o si no existe.

**is\_partition\_of()**

SELECT is\_parent( :child\_schema, :child, :parent\_schema, :parent\_table, :description ); SELECT is\_parent( :child\_schema, :child, :parent\_schema, :parent\_table ); SELECT is\_parent( :child, :parent\_table, :description ); SELECT is\_parent( :child, :parent\_table );

**Parámetros**

:child\_schema

Esquema en el que encontrar la tabla de particiones.

:child

Nombre de una tabla de particiones.

:parent\_schema

Esquema en el que encontrar la tabla particionada.

:parent\_table

Nombre de una tabla particionada.

:description

Una breve descripción de la prueba.

Comprueba que una tabla es una partición de otra tabla. La tabla de particiones (o secundaria) se especifica primero, la tabla de particiones (o matriz) en segundo lugar. Sin los parámetros del esquema, ambas tablas deben estar visibles en la ruta de búsqueda. Si se omite la descripción de la prueba, se establecerá en "La tabla :child\_tabledebe ser una partición de :parent\_table". Tenga en cuenta que esta prueba fallará si alguna de las tablas no existe.

**is\_clustered()**

SELECT is\_clustered( :schema, :table, :index, :description ); SELECT is\_clustered( :schema, :table, :index ); SELECT is\_clustered( :table, :index ); SELECT is\_clustered( :index );

**Parámetros**

:schema

Esquema en el que encontrar la tabla.

:table

Nombre de una tabla que contiene el índice.

:index

Name of the index.

:description

A short description of the test.

Tests whether a table is clustered on the given index. A table is clustered on an index when the SQL command CLUSTER TABLE INDEXNAME has been executed. Clustering reorganizes the table tuples so that they are stored on disk in the order defined by the index.

**is\_indexed()**

SELECT is\_indexed( :schema, :table, :columns, :description ); SELECT is\_indexed( :schema, :table, :columns ); SELECT is\_indexed( :table, :columns, :description ); SELECT is\_indexed( :table, :columns ); SELECT is\_indexed( :schema, :table, :column, :description ); SELECT is\_indexed( :schema, :table, :column ); SELECT is\_indexed( :table, :column, :description ); SELECT is\_indexed( :table, :column );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table containing the index.

:columns

Array of the columns and/or expressions in the index.

:column

Indexed column name or expression.

:description

A short description of the test.

Checks that the specified columns or expressions are contained in a single index on the named table. Effectively like has\_index() except that it doesn't require an index name and does require one or more column names or expressions in the defined for the index.

**index\_is\_type()**

SELECT index\_is\_type( :schema, :table, :index, :type, :description ); SELECT index\_is\_type( :schema, :table, :index, :type ); SELECT index\_is\_type( :table, :index, :type ); SELECT index\_is\_type( :index, :type );

**Parameters**

:schema

Schema in which to find the table.

:table

Name of a table containing the index.

:index

Name of the index.

:type

The index Type.

:description

A short description of the test.

Tests to ensure that an index is of a particular type. At the time of this writing, the supported types are:

* btree
* hash
* gist
* gin

If the test fails, it will emit a diagnostic message with the actual index type, like so:

# Failed test 175: "Index idx\_bar should be a hash index" # have: btree # want: hash

**Feeling Funky**

Perhaps more important than testing the database schema is testing your custom functions. Especially if you write functions that provide the interface for clients to interact with the database, making sure that they work will save you time in the long run. So check out these assertions to maintain your sanity.

**can()**

SELECT can( :schema, :functions, :description ); SELECT can( :schema, :functions ); SELECT can( :functions, :description ); SELECT can( :functions );

**Parameters**

:schema

Schema in which to find the functions.

:functions

Array of function names.

:description

A short description of the test.

Checks to be sure that :schema has :functions defined. This is subtly different from functions\_are() . functions\_are() fails if the functions defined in :schema are not exactly the functions defined in :functions . can() , on the other hand, just makes sure that :functions exist.

If :schema is omitted, then can() will look for functions defined in schemas defined in the search path. No matter how many functions are listed in :functions , a single call to can() counts as one test. If you want otherwise, call can() once for each function — or better yet, use has\_function() . Ejemplo:

SELECT can( 'pg\_catalog', ARRAY['upper', 'lower'] );

If any of the functions are not defined, the test will fail and the diagnostics will output a list of the functions that are missing, like so:

# Failed test 52: "Schema pg\_catalog can" # pg\_catalog.foo() missing # pg\_catalog.bar() missing

**function\_lang\_is()**

SELECT function\_lang\_is( :schema, :function, :args, :language, :description ); SELECT function\_lang\_is( :schema, :function, :args, :language ); SELECT function\_lang\_is( :schema, :function, :language, :description ); SELECT function\_lang\_is( :schema, :function, :language ); SELECT function\_lang\_is( :function, :args, :language, :description ); SELECT function\_lang\_is( :function, :args, :language ); SELECT function\_lang\_is( :function, :language, :description ); SELECT function\_lang\_is( :function, :language );

**Parameters**

:schema

Schema in which to find the function.

:function

Function name.

:args

Array of data types for the function arguments.

:language

Name of the procedural language.

:description

A short description of the test.

Tests that a particular function is implemented in a particular procedural language. The function name is required. If the :schema argument is omitted, then the function must be visible in the search path. If the :args[] argument is passed, then the function with that argument signature will be the one tested; otherwise, a function with any signature will be checked (pass an empty array to specify a function with an empty signature). If the :description is omitted, a reasonable substitute will be created. Ejemplos:

SELECT function\_lang\_is( 'myschema', 'foo', ARRAY['integer', 'text'], 'plperl' ); SELECT function\_lang\_is( 'do\_something', 'sql' ); SELECT function\_lang\_is( 'do\_something', ARRAY['integer'], 'plpgsql' ); SELECT function\_lang\_is( 'do\_something', ARRAY['numeric'], 'plpgsql' );

In the event of a failure, you'll useful diagnostics will tell you what went wrong, for example:

# Failed test 211: "Function mychema.eat(integer, text) should be written in perl" # have: plpgsql # want: perl

If the function does not exist, you'll be told that, too.

# Failed test 212: "Function myschema.grab() should be written in sql" # Function myschema.grab() does not exist

Pero luego verificas has\_function()primero, ¿verdad?

**function\_returns()**

SELECT function\_returns( :schema, :function, :args, :type, :description ); SELECT function\_returns( :schema, :function, :args, :type ); SELECT function\_returns( :schema, :function, :type, :description ); SELECT function\_returns( :schema, :function, :type ); SELECT function\_returns( :function, :args, :type, :description ); SELECT function\_returns( :function, :args, :type ); SELECT function\_returns( :function, :type, :description ); SELECT function\_returns( :function, :type );

**Parámetros**

:schema

Esquema en el que encontrar la función.

:function

Nombre de la función.

:args

Matriz de tipos de datos para los argumentos de la función.

:type

Valor de retorno del tipo de datos.

:description

Una breve descripción de la prueba.

Comprueba que una función particular devuelve un tipo de datos particular. Los argumentos :args[]y :typedeben formatearse como se mostrarían en la vista de una función usando el \dfcomando en psql. Por ejemplo, use "variando el carácter" en lugar de "varchar", y "booleano" en lugar de "bool". Para establecer funciones de retorno, el :typeargumento debe comenzar con "setof" (sí, en minúsculas). Ejemplos:

SELECT function\_returns( 'myschema', 'foo', ARRAY['integer', 'text'], 'integer' ); SELECT function\_returns( 'do\_something', 'setof bool' ); SELECT function\_returns( 'do\_something', ARRAY['integer'], 'boolean' ); SELECT function\_returns( 'do\_something', ARRAY['numeric'], 'numeric' );

Si :schemase omite el argumento, entonces la función debe estar visible en la ruta de búsqueda. Si :args[]se pasa el argumento, entonces la función con esa firma de argumento será la probada; de lo contrario, se comprobará una función con cualquier firma (pase una matriz vacía para especificar una función con una firma vacía). Si :descriptionse omite, se creará un sustituto razonable.

En el caso de una falla, los diagnósticos útiles le dirán qué fue lo que falló, por ejemplo:

# Failed test 283: "Function oww(integer, text) should return integer" # have: bool # want: integer

Si la función no existe, también se lo diremos.

# Failed test 284: "Function oui(integer, text) should return integer" # Function oui(integer, text) does not exist

Pero luego verificas has\_function()primero, ¿verdad?

**is\_definer()**

SELECT is\_definer( :schema, :function, :args, :description ); SELECT is\_definer( :schema, :function, :args ); SELECT is\_definer( :schema, :function, :description ); SELECT is\_definer( :schema, :function ); SELECT is\_definer( :function, :args, :description ); SELECT is\_definer( :function, :args ); SELECT is\_definer( :function, :description ); SELECT is\_definer( :function );

**Parámetros**

:schema

Esquema en el que encontrar la función.

:function

Nombre de la función.

:args

Matriz de tipos de datos para los argumentos de la función.

:description

Una breve descripción de la prueba.

Comprueba que una función es un definidor de seguridad (es decir, una función "setuid"). Si :schemase omite el argumento, entonces la función debe estar visible en la ruta de búsqueda. Si :argsse pasa el argumento, entonces la función con esa firma de argumento será la probada; de lo contrario, se comprobará una función con cualquier firma (pase una matriz vacía para especificar una función con una firma vacía). Si :descriptionse omite, se creará un sustituto razonable. Ejemplos:

SELECT is\_definer( 'myschema', 'foo', ARRAY['integer', 'text'] ); SELECT is\_definer( 'do\_something' ); SELECT is\_definer( 'do\_something', ARRAY['integer'] ); SELECT is\_definer( 'do\_something', ARRAY['numeric'] );

Si la función no existe, un útil mensaje de diagnóstico le informará:

# Failed test 290: "Function nasty() should be security definer" # Function nasty() does not exist

Pero luego verificas has\_function()primero, ¿verdad?

**isnt\_definer()**

SELECT isnt\_definer( :schema, :function, :args, :description ); SELECT isnt\_definer( :schema, :function, :args ); SELECT isnt\_definer( :schema, :function, :description ); SELECT isnt\_definer( :schema, :function ); SELECT isnt\_definer( :function, :args, :description ); SELECT isnt\_definer( :function, :args ); SELECT isnt\_definer( :function, :description ); SELECT isnt\_definer( :function );

**Parámetros**

:schema

Esquema en el que encontrar la función.

:function

Nombre de la función.

:args

Matriz de tipos de datos para los argumentos de la función.

:description

Una breve descripción de la prueba.

This function is the inverse of is\_definer() . The test passes if the specified function is not a security definer.

If the function does not exist, a handy diagnostic message will let you know:

# Failed test 290: "Function nasty() should not be security definer" # Function nasty() does not exist

But then you check with has\_function() first, right?

**is\_strict()**

SELECT is\_strict( :schema, :function, :args, :description ); SELECT is\_strict( :schema, :function, :args ); SELECT is\_strict( :schema, :function, :description ); SELECT is\_strict( :schema, :function ); SELECT is\_strict( :function, :args, :description ); SELECT is\_strict( :function, :args ); SELECT is\_strict( :function, :description ); SELECT is\_strict( :function );

**Parameters**

:schema

Schema in which to find the function.

:function

Function name.

:args

Array of data types for the function arguments.

:description

A short description of the test.

Tests that a function is a strict, meaning that the function returns null if any argument is null. If the :schema argument is omitted, then the function must be visible in the search path. If the :args argument is passed, then the function with that argument signature will be the one tested; otherwise, a function with any signature will be checked (pass an empty array to specify a function with an empty signature). If the :description is omitted, a reasonable substitute will be created. Ejemplos:

SELECT is\_strict( 'myschema', 'foo', ARRAY['integer', 'text'] ); SELECT is\_strict( 'do\_something' ); SELECT is\_strict( 'do\_something', ARRAY['integer'] ); SELECT is\_strict( 'do\_something', ARRAY['numeric'] );

If the function does not exist, a handy diagnostic message will let you know:

# Failed test 290: "Function nasty() should be strict" # Function nasty() does not exist

But then you check with has\_function() first, right?

**isnt\_strict()**

SELECT isnt\_strict( :schema, :function, :args, :description ); SELECT isnt\_strict( :schema, :function, :args ); SELECT isnt\_strict( :schema, :function, :description ); SELECT isnt\_strict( :schema, :function ); SELECT isnt\_strict( :function, :args, :description ); SELECT isnt\_strict( :function, :args ); SELECT isnt\_strict( :function, :description ); SELECT isnt\_strict( :function );

**Parameters**

:schema

Schema in which to find the function.

:function

Function name.

:args

Array of data types for the function arguments.

:description

A short description of the test.

This function is the inverse of is\_strict() . The test passes if the specified function is not strict.

If the function does not exist, a handy diagnostic message will let you know:

# Failed test 290: "Function nasty() should be an aggregate function" # Function nasty() does not exist

But then you check with has\_function() first, right?

**is\_aggregate()**

SELECT is\_aggregate( :schema, :function, :args, :description ); SELECT is\_aggregate( :schema, :function, :args ); SELECT is\_aggregate( :schema, :function, :description ); SELECT is\_aggregate( :schema, :function ); SELECT is\_aggregate( :function, :args, :description ); SELECT is\_aggregate( :function, :args ); SELECT is\_aggregate( :function, :description ); SELECT is\_aggregate( :function );

**Parameters**

:schema

Schema in which to find the function.

:function

Function name.

:args

Array of data types for the function arguments.

:description

A short description of the test.

Tests that a function is an aggregate function. If the :schema argument is omitted, then the function must be visible in the search path. If the :args[] argument is passed, then the function with that argument signature will be the one tested; otherwise, a function with any signature will be checked (pass an empty array to specify a function with an empty signature). If the :description is omitted, a reasonable substitute will be created. Ejemplos:

SELECT is\_aggregate( 'myschema', 'foo', ARRAY['integer', 'text'] ); SELECT is\_aggregate( 'do\_something' ); SELECT is\_aggregate( 'do\_something', ARRAY['integer'] ); SELECT is\_aggregate( 'do\_something', ARRAY['numeric'] );

If the function does not exist, a handy diagnostic message will let you know:

# Failed test 290: "Function nasty() should be an aggregate function" # Function nasty() does not exist

But then you check with has\_function() first, right?

**isnt\_aggregate()**

SELECT isnt\_aggregate( :schema, :function, :args, :description ); SELECT isnt\_aggregate( :schema, :function, :args ); SELECT isnt\_aggregate( :schema, :function, :description ); SELECT isnt\_aggregate( :schema, :function ); SELECT isnt\_aggregate( :function, :args, :description ); SELECT isnt\_aggregate( :function, :args ); SELECT isnt\_aggregate( :function, :description ); SELECT isnt\_aggregate( :function );

**Parameters**

:schema

Schema in which to find the function.

:function

Function name.

:args

Array of data types for the function arguments.

:description

A short description of the test.

Esta función es la inversa de is\_aggregate(). La prueba pasa si la función especificada no es una función agregada.

Si la función no existe, un útil mensaje de diagnóstico le informará:

# Failed test 290: "Function nasty() should not be an aggregate function" # Function nasty() does not exist

Pero luego verificas has\_function()primero, ¿verdad?

**volatility\_is()**

SELECT volatility\_is( :schema, :function, :args, :volatility, :description ); SELECT volatility\_is( :schema, :function, :args, :volatility ); SELECT volatility\_is( :schema, :function, :volatility, :description ); SELECT volatility\_is( :schema, :function, :volatility ); SELECT volatility\_is( :function, :args, :volatility, :description ); SELECT volatility\_is( :function, :args, :volatility ); SELECT volatility\_is( :function, :volatility, :description ); SELECT volatility\_is( :function, :volatility );

**Parámetros**

:schema

Esquema en el que encontrar la función.

:function

Nombre de la función.

:args

Matriz de tipos de datos para los argumentos de la función.

:volatility

Nivel de volatilidad.

:description

Una breve descripción de la prueba.

Prueba la volatilidad de una función. Las volatilidades soportadas son "volátiles", "estables" e "inmutables". Consulte la [CREATE FUNCTIONdocumentación](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.postgresql.org/docs/current/static/sql-createfunction.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjhM3TygW0v33AvEx0jDqm6t-3mlw) para más detalles. El nombre de la función es obligatorio. Si :schemase omite el argumento, entonces la función debe estar visible en la ruta de búsqueda. Si :args[]se pasa el argumento, entonces la función con esa firma de argumento será la probada; de lo contrario, se comprobará una función con cualquier firma (pase una matriz vacía para especificar una función con una firma vacía). Si :descriptionse omite, se creará un sustituto razonable. Ejemplos:

SELECT volatility\_is( 'myschema', 'foo', ARRAY['integer', 'text'], 'stable' ); SELECT volatility\_is( 'do\_something', 'immutable' ); SELECT volatility\_is( 'do\_something', ARRAY['integer'], 'stable' ); SELECT volatility\_is( 'do\_something', ARRAY['numeric'], 'volatile' );

En el caso de una falla, los diagnósticos útiles le dirán qué fue lo que falló, por ejemplo:

# Failed test 211: "Function mychema.eat(integer, text) should be IMMUTABLE" # have: VOLATILE # want: IMMUTABLE

Si la función no existe, también se lo diremos.

# Failed test 212: "Function myschema.grab() should be IMMUTABLE" # Function myschema.grab() does not exist

Pero luego verificas has\_function()primero, ¿verdad?

**trigger\_is()**

SELECT trigger\_is( :schema, :table, :trigger, :func\_schema, :function, :description ); SELECT trigger\_is( :schema, :table, :trigger, :func\_schema, :function ); SELECT trigger\_is( :table, :trigger, :function, :description ); SELECT trigger\_is( :table, :trigger, :function );

**Parámetros**

:schema

Schema in which to find the table.

:table

Table in which to find the trigger.

:trigger

Trigger name.

:func\_schema

Schema in which to find the trigger function.

:function

Function name.

:description

A short description of the test.

Tests that the specified trigger calls the named function. If not, it outputs a useful diagnostic:

# Failed test 31: "Trigger set\_users\_pass should call hash\_password()" # have: hash\_pass # want: hash\_password

**Database Deets**

Tables and functions aren't the only objects in the database, as you well know. These assertions close the gap by letting you test the attributes of other database objects.

**language\_is\_trusted()**

SELECT language\_is\_trusted( language, description ); SELECT language\_is\_trusted( language );

**Parameters**

:language

Name of a procedural language.

:description

A short description of the test.

Tests that the specified procedural language is trusted. See the [CREATE LANGUAGE](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.postgresql.org/docs/current/static/sql-createlanguage.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhgPWnayg870RBwAwMg-4_quvJUHQA) documentation for details on trusted and untrusted procedural languages. If the :description argument is not passed, a suitably useful default will be created.

In the event that the language in question does not exist in the database, language\_is\_trusted() will emit a diagnostic message to alert you to this fact, like so:

# Failed test 523: "Procedural language plomgwtf should be trusted" # Procedural language plomgwtf does not exist

But you really ought to call has\_language() first so that you never get that far.

**enum\_has\_labels()**

SELECT enum\_has\_labels( :schema, :enum, :labels, :description ); SELECT enum\_has\_labels( :schema, :enum, :labels ); SELECT enum\_has\_labels( :enum, :labels, :description ); SELECT enum\_has\_labels( :enum, :labels );

**Parameters**

:schema

Schema in which to find the enum.

:enum

Enum name.

:labels

An array of the enum labels.

:description

A short description of the test.

This function tests that an enum consists of an expected list of labels. Enums are supported in PostgreSQL 8.3 or higher. The first argument is a schema name, the second an enum name, the third an array of enum labels, and the fourth a description. Ejemplo:

SELECT enum\_has\_labels( 'myschema', 'someenum', ARRAY['foo', 'bar'] );

If you omit the schema, the enum must be visible in the search path. If you omit the test description, it will be set to “Enum :enum should have labels ( :labels )”.

**domain\_type\_is()**

SELECT domain\_type\_is( :schema, :domain, :type\_schema, :type, :description ); SELECT domain\_type\_is( :schema, :domain, :type\_schema, :type ); SELECT domain\_type\_is( :schema, :domain, :type, :description ); SELECT domain\_type\_is( :schema, :domain, :type ); SELECT domain\_type\_is( :domain, :type, :description ); SELECT domain\_type\_is( :domain, :type );

**Parameters**

:schema

Schema in which to find the domain.

:domain

Nombre de dominio.

:type\_schema

Esquema en el que encontrar el tipo de datos.

:type

Tipo de datos del dominio.

:description

Una breve descripción de la prueba.

Prueba el tipo de datos subyacente a un dominio. Los dos primeros argumentos son el esquema y el nombre del dominio. Los dos segundos son el esquema y el nombre del tipo que el dominio debe extender. El quinto argumento es una descripción. Si no hay una descripción, se creará una descripción predeterminada razonable.

Los argumentos del esquema también son opcionales. Sin embargo, si no hay un :schemaargumento, no puede haber un :type\_schemaargumento, tampoco, aunque el esquema se puede incluir en el typeargumento, por ejemplo, contrib.citext.

Para las formas de 3 y 4 argumentos con esquemas, convierta los esquemas NAMEpara evitar ambigüedades. Ejemplo:

SELECT domain\_type\_is( 'public'::name, 'us\_postal\_code', 'public'::name, 'text' );

Si el tipo de datos no coincide con el tipo que se extiende el dominio, la prueba fallará y los diagnósticos de salida serán así:

# Failed test 631: "Domain public.us\_postal\_code should extend type public.integer" # have: public.text # want: public.integer

Si el dominio en cuestión no existe realmente, la prueba fallará con los diagnósticos que lo indiquen:

# Failed test 632: "Domain public.zip\_code should extend type public.text" # Domain public.zip\_code does not exist

**domain\_type\_isnt()**

SELECT domain\_type\_isnt( :schema, :domain, :type\_schema, :type, :description ); SELECT domain\_type\_isnt( :schema, :domain, :type\_schema, :type ); SELECT domain\_type\_isnt( :schema, :domain, :type, :description ); SELECT domain\_type\_isnt( :schema, :domain, :type ); SELECT domain\_type\_isnt( :domain, :type, :description ); SELECT domain\_type\_isnt( :domain, :type );

**Parámetros**

:schema

Esquema en el que encontrar el dominio.

:domain

Nombre de dominio.

:type\_schema

Esquema en el que encontrar el tipo de datos.

:type

Tipo de datos del dominio.

:description

Una breve descripción de la prueba.

A la inversa de domain\_type\_is(), esta función prueba que un dominio *no* extiende un tipo de datos en particular. Por ejemplo, un dominio de código postal de EE. UU. Probablemente debería extender el texttipo, no integer, ya que los 0 iniciales son válidos y requeridos. Ejemplo:

SELECT domain\_type\_isnt( 'public', 'us\_postal\_code', 'public', 'integer', 'The us\_postal\_code domain should not extend the integer type' );

Los argumentos son los mismos que para domain\_type\_is().

**cast\_context\_is()**

SELECT cast\_context\_is( :source\_type, :target\_type, :context, :description ); SELECT cast\_context\_is( :source\_type, :target\_type, :context );

**Parámetros**

:source\_type

El tipo emitido desde.

:target\_type

El tipo emitido a.

:context

El contexto para el reparto, uno de "implícito", "asignación" o "explícito".

Test that a cast from a source to a target data type has a particular context. Ejemplo:

SELECT cast\_context\_is( 'integer', 'bigint', 'implicit' );

The data types should be passed as they are displayed by pg\_catalog.format\_type() . For example, you would need to pass “character varying”, and not “VARCHAR”.

The The supported contexts are “implicit”, “assignment”, and “explicit”. You can also just pass in “i”, “a”, or “e”. Consult the PostgreSQL [CREATE CAST](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.postgresql.org/docs/current/static/sql-createcast.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhhEIgQccRseeTj8QAuj9aaIEd8PvQ) documentation for the differences between these contexts (hint: they correspond to the default context, AS IMPLICIT , and AS ASSIGNMENT ). If you don't supply a test description, pgTAP will create a reasonable one for you.

Test failure will result in useful diagnostics, such as:

# Failed test 124: "Cast ("integer" AS "bigint") context should be explicit" # have: implicit # want: explicit

If the cast doesn't exist, you'll be told that, too:

# Failed test 199: "Cast ("integer" AS foo) context should be explicit" # Cast ("integer" AS foo) does not exist

But you've already used has\_cast() to make sure of that, right?

**is\_superuser()**

SELECT is\_superuser( :user, :description ); SELECT is\_superuser( :user );

**Parameters**

:user

Nombre de un usuario de PostgreSQL.

:description

Una breve descripción de la prueba.

Comprueba que un usuario de la base de datos es un superusuario. Si se omite la descripción, se establecerá de forma predeterminada en "El usuario :userdebe ser un superusuario". Ejemplo:

SELECT is\_superuser('theory' ;

Si el usuario no existe en la base de datos, los diagnósticos lo indicarán.

**isnt\_superuser()**

SELECT is\_superuser( 'dr\_evil', 'User "dr\_evil" should not be a super user' );

**Parámetros**

:user

Nombre de un usuario de PostgreSQL.

:description

Una breve descripción de la prueba.

A la inversa de is\_superuser(), esta función prueba que un usuario de la base de datos *no* es un superusuario. Tenga en cuenta que si el usuario nombrado no existe en la base de datos, la prueba aún se considera un error y los diagnósticos lo indicarán.

**is\_member\_of()**

SELECT is\_member\_of( :role, :members, :description ); SELECT is\_member\_of( :role, :members ); SELECT is\_member\_of( :role, :member, :description ); SELECT is\_member\_of( :role, :member );

**Parámetros**

:role

Nombre de un rol de grupo PostgreSQL.

:members

Conjunto de nombres de roles que deben ser miembros del rol de grupo.

:member

Nombre de un rol que debe ser miembro del rol de grupo.

:description

Una breve descripción de la prueba.

SELECCIONA is\_member\_of ('sweeties', 'anna' 'Anna debería ser una dulzura'); SELECT is\_member\_of ('meanies', ARRAY ['dr\_evil', 'dr\_no']);

Comprueba si un rol de grupo contiene un rol de miembro o todo un conjunto de roles de miembros. Si se omite la descripción, se establecerá de manera predeterminada "Debería tener miembros de rol :role". En caso de error, se is\_member\_of()mostrarán los diagnósticos que enumeran los roles de miembro faltantes, como por ejemplo:

# Failed test 370: "Should have members of role meanies" # Members missing from the meanies role: # theory # agliodbs

Si el rol de grupo no existe, los diagnósticos le indicarán eso, en su lugar. Pero usas has\_role()para asegurarte de que el rol existe antes de verificar sus miembros, ¿no es así? Por supuesto que sí.

**rule\_is\_instead()**

SELECT rule\_is\_instead( :schema, :table, :rule, :description ); SELECT rule\_is\_instead( :schema, :table, :rule ); SELECT rule\_is\_instead( :table, :rule, :description ); SELECT rule\_is\_instead( :table, :rule );

**Parámetros**

:schema

Nombre de un esquema en el que se encuentra la tabla.

:table

Nombre de la tabla a la que se aplica la regla.

:rule

Un nombre de regla.

:description

Una breve descripción de la prueba.

Checks whether a rule on the specified relation is an INSTEAD rule. See the [CREATE RULE Documentation](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.postgresql.org/docs/current/static/sql-createrule.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhh7r2FP0tMywgEOJlDZhRluJYUXBw) for details. If the :schema argument is omitted, the relation must be visible in the search path. If the :description argument is omitted, an appropriate description will be created. An example:

SELECT rule\_is\_instead('public', 'users', 'on\_insert');

In the event that the test fails because the rule in question does not actually exist, you will see an appropriate diagnostic such as:

# Failed test 625: "Rule on\_insert on relation public.users should be an INSTEAD rule" # Rule on\_insert does not exist

**rule\_is\_on()**

SELECT rule\_is\_on( :schema, :table, :rule, :event, :description ); SELECT rule\_is\_on( :schema, :table, :rule, :event ); SELECT rule\_is\_on( :table, :rule, :event, :description ); SELECT rule\_is\_on( :table, :rule, :event );

**Parameters**

:schema

Name of a schema in which to find the table.

:table

Name of the table to which the rule is applied.

:rule

A rule name.

:event

Name of a rule event, one of “SELECT”, “INSERT”, “UPDATE”, or “DELETE”.

:description

A short description of the test.

Prueba el evento en busca de una regla, que puede ser una de "SELECT", "INSERT", "UPDATE", o "DELETE". Para el :eventargumento, puede especificar el nombre del evento en cualquier caso, o incluso con una sola letra ("s", "i", "u" o "d"). Si :schemase omite el argumento, la tabla debe estar visible en la ruta de búsqueda. Si :descriptionse omite, se creará un valor predeterminado razonable. Ejemplo:

SELECT rule\_is\_on('public', 'users', 'on\_insert', 'INSERT');

Si la prueba falla, verá diagnósticos útiles, como:

# Failed test 133: "Rule ins\_me should be on INSERT to public.widgets" # have: UPDATE # want: INSERT

Si la regla en cuestión no existe, también se le dirá que:

# Failed test 134: "Rule upd\_me should be on UPDATE to public.widgets" # Rule upd\_me does not exist on public.widgets

Pero luego corres has\_rule()primero, ¿no?

**¿Quién me posee?**

Después de probar la disponibilidad de varios objetos, a menudo necesitamos saber quién es el propietario de un objeto.

**db\_owner\_is ()**

SELECT db\_owner\_is ( :dbname, :user, :description ); SELECT db\_owner\_is ( :dbname, :user );

**Parámetros**

:dbname

Nombre de una base de datos.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de la base de datos. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT db\_owner\_is( 'mydb', 'someuser', 'mydb should be owned by someuser' ); SELECT db\_owner\_is( current\_database(), current\_user );

En el caso de que la prueba falle porque la base de datos en cuestión no existe realmente, verá un diagnóstico apropiado como:

# Failed test 16: "Database foo should be owned by www" # Database foo does not exist

Si la prueba falla porque la base de datos no es propiedad del usuario especificado, los diagnósticos tendrán un aspecto similar al siguiente:

# Failed test 17: "Database bar should be owned by root" # have: postgres # want: root

**schema\_owner\_is ()**

SELECT schema\_owner\_is ( :schemaname, :user, :description ); SELECT schema\_owner\_is ( :schemaname, :user );

**Parámetros**

:schemaname

Nombre de un esquema.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad del esquema. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT schema\_owner\_is( 'myschema', 'someuser', 'myschema should be owned by someuser' ); SELECT schema\_owner\_is( current\_schema(), current\_user );

En el caso de que la prueba falle porque el esquema en cuestión no existe realmente, verá un diagnóstico apropiado como:

# Failed test 16: "Schema foo should be owned by www" # Schema foo does not exist

Si la prueba falla porque el esquema no es propiedad del usuario especificado, los diagnósticos tendrán un aspecto similar al siguiente:

# Failed test 17: "Schema bar should be owned by root" # have: postgres # want: root

**tablespace\_owner\_is ()**

SELECT tablespace\_owner\_is ( :tablespacename, :user, :description ); SELECT tablespace\_owner\_is ( :tablespacename, :user );

**Parámetros**

:tablespacename

Nombre de un tablespace.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad del tablespace. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT tablespace\_owner\_is( 'myts', 'joe', 'Joe has mytablespace' ); SELECT tablespace\_owner\_is( 'pg\_default', current\_user );

En el caso de que la prueba falle porque el espacio de tablas en cuestión no existe realmente, verá un diagnóstico apropiado como:

# Failed test 16: "Tablespace ssd should be owned by www" # Tablespace ssd does not exist

Si la prueba falla porque el espacio de tablas no es propiedad del usuario especificado, los diagnósticos tendrán un aspecto similar al siguiente:

# Failed test 17: "Tablespace raid\_hds should be owned by root" # have: postgres # want: root

**relation\_owner\_is ()**

SELECT relation\_owner\_is ( :schema, :relation, :user, :description ); SELECT relation\_owner\_is ( :relation, :user, :description ); SELECT relation\_owner\_is ( :schema, :relation, :user ); SELECT relation\_owner\_is ( :relation, :user );

**Parámetros**

:schema

Nombre de un esquema en el que buscar el :relation.

:relation

Nombre de una relación.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de una relación. Las relaciones son tablas, vistas, vistas materializadas, secuencias, tipos compuestos, tablas externas y tablas de brindis. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT relation\_owner\_is( 'public', 'mytable', 'someuser', 'mytable should be owned by someuser' ); SELECT relation\_owner\_is( current\_schema(), 'mysequence', current\_user );

In the event that the test fails because the relation in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Relation foo should be owned by www" # Relation foo does not exist

If the test fails because the relation is not owned by the specified user, the diagnostics will look something like:

# Failed test 17: "Relation bar should be owned by root" # have: postgres # want: root

**table\_owner\_is ()**

SELECT table\_owner\_is ( :schema, :table, :user, :description ); SELECT table\_owner\_is ( :table, :user, :description ); SELECT table\_owner\_is ( :schema, :table, :user ); SELECT table\_owner\_is ( :table, :user );

**Parameters**

:schema

Name of a schema in which to find the :table .

:table

Name of a table.

:user

Name of a user.

:description

A short description of the test.

Tests the ownership of a table. If the :description argument is omitted, an appropriate description will be created. Ejemplos:

SELECT table\_owner\_is( 'public', 'mytable', 'someuser', 'mytable should be owned by someuser' ); SELECT table\_owner\_is( 'widgets', current\_user );

Note that this function will not recognize foreign tables; use foreign\_table\_owner\_is() to test for the presence of foreign tables.

In the event that the test fails because the table in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Table foo should be owned by www" # Table foo does not exist

Si la prueba falla porque la tabla no es propiedad del usuario especificado, los diagnósticos se verán como:

# Failed test 17: "Table bar should be owned by root" # have: postgres # want: root

**view\_owner\_is ()**

SELECT view\_owner\_is ( :schema, :view, :user, :description ); SELECT view\_owner\_is ( :view, :user, :description ); SELECT view\_owner\_is ( :schema, :view, :user ); SELECT view\_owner\_is ( :view, :user );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :view.

:view

Nombre de una vista.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de una vista. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT view\_owner\_is( 'public', 'myview', 'someuser', 'myview should be owned by someuser' ); SELECT view\_owner\_is( 'widgets', current\_user );

En el caso de que la prueba falle porque la vista en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "View foo should be owned by www" # View foo does not exist

Si la prueba falla porque la vista no es propiedad del usuario especificado, los diagnósticos se verán como:

# Failed test 17: "View bar should be owned by root" # have: postgres # want: root

**materialized\_view\_owner\_is ()**

SELECT materialized\_view\_owner\_is ( :schema, :materialized\_view, :user, :description ); SELECT materialized\_view\_owner\_is ( :materialized\_view, :user, :description ); SELECT materialized\_view\_owner\_is ( :schema, :materialized\_view, :user ); SELECT materialized\_view\_owner\_is ( :materialized\_view, :user );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :materialized\_view.

:materialized\_view

Nombre de una vista materializada.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de una vista materializada. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT view\_owner\_is( 'public', 'my\_matview', 'someuser', 'my\_matview should be owned by someuser' ); SELECT materialized\_view\_owner\_is( 'widgets', current\_user );

En el caso de que la prueba falle porque la vista materializada en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "Materialized view foo should be owned by www" # Materialized view foo does not exist

Si la prueba falla porque la vista materializada no es propiedad del usuario especificado, los diagnósticos se verán como:

# Failed test 17: "Materialized view bar should be owned by root" # have: postgres # want: root

**sequence\_owner\_is ()**

SELECT sequence\_owner\_is ( :schema, :sequence, :user, :description ); SELECT sequence\_owner\_is ( :sequence, :user, :description ); SELECT sequence\_owner\_is ( :schema, :sequence, :user ); SELECT sequence\_owner\_is ( :sequence, :user );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :sequence.

:sequence

Nombre de una secuencia.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de una secuencia. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT sequence\_owner\_is( 'public', 'mysequence', 'someuser', 'mysequence should be owned by someuser' ); SELECT sequence\_owner\_is( 'widgets', current\_user );

En el caso de que la prueba falle porque la secuencia en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "Sequence foo should be owned by www" # Sequence foo does not exist

Si la prueba falla porque la secuencia no es propiedad del usuario especificado, los diagnósticos tendrán un aspecto similar al siguiente:

# Failed test 17: "Sequence bar should be owned by root" # have: postgres # want: root

**composite\_owner\_is ()**

SELECT composite\_owner\_is ( :schema, :composite, :user, :description ); SELECT composite\_owner\_is ( :composite, :user, :description ); SELECT composite\_owner\_is ( :schema, :composite, :user ); SELECT composite\_owner\_is ( :composite, :user );

**Parámetros**

:schema

Nombre de un esquema en el que se encuentra el :compositetipo.

:composite

Nombre de un tipo compuesto.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de un compuesto. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT composite\_owner\_is( 'public', 'mycomposite', 'someuser', 'mycomposite should be owned by someuser' ); SELECT composite\_owner\_is( 'widgets', current\_user );

En el caso de que la prueba falle porque el compuesto en cuestión no existe realmente o no es visible, verá un diagnóstico apropiado como:

# Failed test 16: "Composite type foo should be owned by www" # Composite type foo does not exist

Si la prueba falla porque el compuesto no es propiedad del usuario especificado, los diagnósticos tendrán un aspecto similar al siguiente:

# Failed test 17: "Composite type bar should be owned by root" # have: postgres # want: root

**foreign\_table\_owner\_is ()**

SELECT foreign\_table\_owner\_is ( :schema, :foreign\_table, :user, :description ); SELECT foreign\_table\_owner\_is ( :foreign\_table, :user, :description ); SELECT foreign\_table\_owner\_is ( :schema, :foreign\_table, :user ); SELECT foreign\_table\_owner\_is ( :foreign\_table, :user );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :foreign\_table.

:foreign\_table

Nombre de una mesa extranjera.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de una mesa extranjera. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT foreign\_table\_owner\_is( 'public', 'mytable', 'someuser', 'mytable should be owned by someuser' ); SELECT foreign\_table\_owner\_is( 'widgets', current\_user );

En el caso de que la prueba falle porque la tabla externa en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "Foreign table foo should be owned by www" # Foreign table foo does not exist

Si la prueba falla porque la tabla externa no es propiedad del usuario especificado, los diagnósticos se verán como:

# Failed test 17: "Foreign table bar should be owned by root" # have: postgres # want: root

**index\_owner\_is ()**

SELECT index\_owner\_is ( :schema, :table, :index, :user, :description ); SELECT index\_owner\_is ( :table, :index, :user, :description ); SELECT index\_owner\_is ( :schema, :table, :index, :user ); SELECT index\_owner\_is ( :table, :index, :user );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :table, :index.

:table

Nombre de una tabla.

:table

Nombre de un índice en la tabla.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de un índice. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT index\_owner\_is( 'public', 'mytable', 'idx\_name', 'someuser', 'Index "idx\_name" on mytable should be owned by someuser' ); SELECT index\_owner\_is( 'widgets', 'widgets\_pkey', current\_user );

En el caso de que la prueba falle porque el índice en cuestión no existe realmente, o si la tabla o el esquema en que se encuentra no existe o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "Index idx\_foo should be owned by root" # Index idx\_foo on table darfoo not found

Si la prueba falla porque la tabla no es propiedad del usuario especificado, los diagnósticos se verán como:

# Failed test 17: "Index idx\_foo on table bar should be owned by bob" # have: postgres # want: bob

**function\_owner\_is ()**

SELECT function\_owner\_is ( :schema, :function, :args, :user, :description ); SELECT function\_owner\_is ( :function, :args, :user, :description ); SELECT function\_owner\_is ( :schema, :function, :args, :user ); SELECT function\_owner\_is ( :function, :args, :user );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :function.

:function

Nombre de una función.

:args

Matriz de tipos de datos de los argumentos de la función.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de una función. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT function\_owner\_is( 'public', 'frobulate', ARRAY['integer', 'text'], 'someuser', 'public.frobulate(integer, text) should be owned by someuser' ); SELECT function\_owner\_is( 'masticate', ARRAY['text'], current\_user );

En el caso de que la prueba falle porque la función en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "Function foo() should be owned by www" # Function foo() does not exist

Si la prueba falla porque la función no es propiedad del usuario especificado, los diagnósticos tendrán un aspecto similar al siguiente:

# Failed test 17: "Function bar() should be owned by root" # have: postgres # want: root

**language\_owner\_is ()**

SELECT language\_owner\_is ( :languagename, :user, :description ); SELECT language\_owner\_is ( :languagename, :user );

**Parámetros**

:languagename

Nombre de un idioma.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de un lenguaje procesal. Si :descriptionse omite el argumento, se creará una descripción apropiada. Funciona en PostgreSQL 8.3 y superior. Ejemplos:

SELECT language\_owner\_is( 'plpgsql', 'larry', 'Larry should own plpgsql' ); SELECT language\_owner\_is( 'plperl', current\_user );

En el caso de que la prueba falle porque el idioma en cuestión no existe realmente, verá un diagnóstico apropiado como:

# Failed test 16: "Language pllolcode should be owned by meow" # Language pllolcode does not exist

Si la prueba falla porque el idioma no es propiedad del usuario especificado, los diagnósticos se verán como:

# Failed test 17: "Language plruby should be owned by mats" # have: postgres # want: mats

**opclass\_owner\_is ()**

SELECT opclass\_owner\_is ( :schema, :opclass, :user, :description ); SELECT opclass\_owner\_is ( :opclass, :user, :description ); SELECT opclass\_owner\_is ( :schema, :opclass, :user ); SELECT opclass\_owner\_is ( :opclass, :user );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :opclass.

:opclass

Nombre de una clase de operador.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de una clase de operador. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT opclass\_owner\_is( 'pg\_catalog', 'int4\_ops', 'postgres', 'Operator class int4\_ops should be owned by postgres' ); SELECT opclass\_owner\_is( 'my\_ops', current\_user );

En el caso de que la prueba falle porque la clase de operador en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "operator class foo should be owned by www" # operator class foo does not exist

Si la prueba falla porque la clase del operador no es propiedad del usuario especificado, los diagnósticos se verán como:

# Failed test 17: "operator class bar should be owned by root" # have: postgres # want: root

**type\_owner\_is ()**

SELECT type\_owner\_is ( :schema, :type, :user, :description ); SELECT type\_owner\_is ( :type, :user, :description ); SELECT type\_owner\_is ( :schema, :type, :user ); SELECT type\_owner\_is ( :type, :user );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :type.

:type

Nombre de un tipo.

:user

Nombre de un usuario.

:description

Una breve descripción de la prueba.

Prueba la propiedad de un tipo de datos. Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT type\_owner\_is( 'pg\_catalog', 'int4', 'postgres', 'type int4 should be owned by postgres' ); SELECT type\_owner\_is( 'us\_postal\_code', current\_user );

En el caso de que la prueba falle porque el tipo en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "type uk\_postal\_code should be owned by www" # type uk\_postal\_code does not exist

Si la prueba falla porque el tipo no es propiedad del usuario especificado, los diagnósticos serán similares a los siguientes:

# Failed test 17: "type us\_postal\_code should be owned by root" # have: postgres # want: root

**Acceso privilegiado**

Entonces sabemos quién es el dueño de los objetos. Pero ¿qué pasa con otros roles? ¿Pueden acceder a los objetos de la base de datos? ¡Vamos a averiguar!

**database\_privs\_are()**

SELECT database\_privs\_are ( :db, :role, :privileges, :description ); SELECT database\_privs\_are ( :db, :role, :privileges );

**Parámetros**

:db

Nombre de una base de datos.

:role

Nombre de un usuario o rol de grupo.

:privileges

Una matriz de privilegios de tabla que el rol debe otorgarse a la base de datos.

:description

Una breve descripción de la prueba.

Prueba los privilegios otorgados a un rol para acceder a una base de datos. Los privilegios de base de datos disponibles son:

* CREAR
* CONECTAR
* TEMPORAL

Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT database\_privs\_are( 'flipr', 'fred', ARRAY['CONNECT', 'TEMPORARY'], 'Fred should be granted CONNECT and TERMPORARY on db "flipr"' ); SELECT database\_privs\_are( 'dept\_corrections', ARRAY['CREATE'] );

Si a la función se le otorgan permisos distintos a los especificados, los diagnósticos enumerarán los permisos adicionales, como así:

# Failed test 14: "Role bob should be granted CREATE on database banks" # Extra privileges: # CONNECT # TEMPORARY

Del mismo modo, si a la función no se le otorgan algunos de los permisos especificados en la base de datos:

# Failed test 15: "Role kurk should be granted CONNECT, TEMPORARY on database banks" # Missing privileges: # CREATE

En el caso de que la prueba falle porque la base de datos en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "Role slim should be granted CONNECT on database maindb" # Database maindb does not exist

If the test fails because the role does not exist, the diagnostics will look something like:

# Failed test 17: "Role slim should be granted CONNECT, CREATE on database widgets" # Role slim does not exist

**tablespace\_privs\_are()**

SELECT tablespace\_privs\_are ( :tablespace, :role, :privileges, :description ); SELECT tablespace\_privs\_are ( :tablespace, :role, :privileges );

**Parameters**

:tablespace

Name of a tablespace.

:role

Name of a user or group role.

:privileges

An array of table privileges the role should be granted to the tablespace.

:description

A short description of the test.

Tests the privileges granted to a role to access a tablespace. The available function privileges are:

* CREAR

If the :description argument is omitted, an appropriate description will be created. Ejemplos:

SELECT tablespace\_privs\_are( 'ssd', 'fred', ARRAY['CREATE'], 'Fred should be granted CREATE on tablespace "ssd"' ); SELECT tablespace\_privs\_are( 'san', ARRAY['CREATE'] );

If the role is granted permissions other than those specified, the diagnostics will list the extra permissions, like so:

# Failed test 14: "Role bob should be granted no privileges on tablespace hdd" # Extra privileges: # CREATE

Likewise if the role is not granted some of the specified permissions on the tablespace:

# Failed test 15: "Role kurk should be granted USAGE on ssd" # Missing privileges: # CREATE

In the event that the test fails because the tablespace in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Role slim should be granted CREATE on tablespace tape" # Tablespace tape does not exist

If the test fails because the role does not exist, the diagnostics will look something like:

# Failed test 17: "Role slim should be granted CREATE on san" # Role slim does not exist

**schema\_privs\_are()**

SELECT schema\_privs\_are ( :schema, :role, :privileges, :description ); SELECT schema\_privs\_are ( :schema, :role, :privileges );

**Parameters**

:schema

Name of a schema.

:role

Name of a user or group role.

:privileges

An array of table privileges the role should be granted to the schema.

:description

A short description of the test.

Tests the privileges granted to a role to access a schema. The available schema privileges are:

* CREAR
* USAGE

If the :description argument is omitted, an appropriate description will be created. Ejemplos:

SELECT schema\_privs\_are( 'flipr', 'fred', ARRAY['CREATE', 'USAGE'], 'Fred should be granted CREATE and USAGE on schema "flipr"' ); SELECT schema\_privs\_are( 'hr', ARRAY['USAGE'] );

If the role is granted permissions other than those specified, the diagnostics will list the extra permissions, like so:

# Failed test 14: "Role bob should be granted no privileges on schema pinata" # Extra privileges: # CREATE # USAGE

Likewise if the role is not granted some of the specified permissions on the schema:

# Failed test 15: "Role kurk should be granted CREATE, USAGE on schema stuff" # Missing privileges: # CREATE

In the event that the test fails because the schema in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Role slim should be granted USAGE on schema main" # Schema main does not exist

If the test fails because the role does not exist, the diagnostics will look something like:

# Failed test 17: "Role slim should be granted CREATE, USAGE on schema admin" # Role slim does not exist

**table\_privs\_are()**

SELECT table\_privs\_are ( :schema, :table, :role, :privileges, :description ); SELECT table\_privs\_are ( :schema, :table, :role, :privileges ); SELECT table\_privs\_are ( :table, :role, :privileges, :description ); SELECT table\_privs\_are ( :table, :role, :privileges );

**Parameters**

:schema

Name of a schema in which to find the table.

:table

Name of a table.

:role

Name of a user or group role.

:privileges

An array of table privileges the role should be granted to the table.

:description

A short description of the test.

Tests the privileges granted to a role to access a table. The available table privileges are:

* DELETE
* INSERT
* Referencias
* RULE
* SELECCIONAR
* TRIGGER
* TRUNCATE
* UPDATE

Note that the privilege RULE is not available after PostgreSQL 8.1, and that TRIGGER was added in 8.4.

If the :description argument is omitted, an appropriate description will be created. Ejemplos:

SELECT table\_privs\_are( 'public', 'frobulate', 'fred', ARRAY['SELECT', 'DELETE'], 'Fred should be able to select and delete on frobulate' ); SELECT table\_privs\_are( 'widgets', 'slim', ARRAY['INSERT', 'UPDATE'] );

If the role is granted permissions other than those specified, the diagnostics will list the extra permissions, like so:

# Failed test 14: "Role bob should be granted SELECT on widgets" # Extra privileges: # DELETE # INSERT # UPDATE

Likewise if the role is not granted some of the specified permissions on the table:

# Failed test 15: "Role kurk should be granted SELECT, INSERT, UPDATE on widgets" # Missing privileges: # INSERT # UPDATE

In the event that the test fails because the table in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Role slim should be granted SELECT on widgets" # Table widgets does not exist

If the test fails because the role does not exist, the diagnostics will look something like:

# Failed test 17: "Role slim should be granted SELECT on widgets" # Role slim does not exist

**sequence\_privs\_are()**

SELECT sequence\_privs\_are ( :schema, :sequence, :role, :privileges, :description ); SELECT sequence\_privs\_are ( :schema, :sequence, :role, :privileges ); SELECT sequence\_privs\_are ( :sequence, :role, :privileges, :description ); SELECT sequence\_privs\_are ( :sequence, :role, :privileges );

**Parameters**

:schema

Name of a schema in which to find the sequence.

:sequence

Name of a sequence.

:role

Name of a user or group role.

:privileges

An array of sequence privileges the role should be granted to the sequence.

:description

A short description of the test.

Tests the privileges granted to a role to access a sequence. The available sequence privileges are:

* SELECCIONAR
* UPDATE
* USAGE

Note that sequence privileges were added in PostgreSQL 9.0, so this function will likely throw an exception on earlier versions.

If the :description argument is omitted, an appropriate description will be created. Ejemplos:

SELECT sequence\_privs\_are( 'public', 'seq\_ids', 'fred', ARRAY['SELECT', 'UPDATE'], 'Fred should be able to select and update seq\_ids' ); SELECT sequence\_privs\_are( 'seq\_u', 'slim', ARRAY['USAGE'] );

If the role is granted permissions other than those specified, the diagnostics will list the extra permissions, like so:

# Failed test 14: "Role bob should be granted SELECT on seq\_foo\_id" # Extra privileges: # UPDATE # USAGE

Likewise if the role is not granted some of the specified permissions on the sequence:

# Failed test 15: "Role kurk should be granted USAGE on seq\_widgets" # Missing privileges: # SELECT # UPDATE

In the event that the test fails because the sequence in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Role slim should be granted SELECT on seq\_widgets" # Sequence widgets does not exist

If the test fails because the role does not exist, the diagnostics will look something like:

# Failed test 17: "Role slim should be granted SELECT on seq\_widgets" # Role slim does not exist

**any\_column\_privs\_are()**

SELECT any\_column\_privs\_are ( :schema, :table, :role, :privileges, :description ); SELECT any\_column\_privs\_are ( :schema, :table, :role, :privileges ); SELECT any\_column\_privs\_are ( :table, :role, :privileges, :description ); SELECT any\_column\_privs\_are ( :table, :role, :privileges );

**Parameters**

:schema

Name of a schema in which to find the table.

:table

Name of a table.

:role

Name of a user or group role.

:privileges

An array of table privileges the role should be granted to the table.

:description

A short description of the test.

Tests the privileges granted to access one or more of the columns in a table. The available column privileges are:

* INSERT
* Referencias
* SELECCIONAR
* UPDATE

Note that column privileges were added in PostgreSQL 8.4, so this function will likely throw an exception on earlier versions.

If the :description argument is omitted, an appropriate description will be created. Ejemplos:

SELECT any\_column\_privs\_are( 'public', 'frobulate', 'fred', ARRAY['SELECT', 'UPDATE'], 'Fred should be able to select and update columns in frobulate' ); SELECT any\_column\_privs\_are( 'widgets', 'slim', ARRAY['INSERT', 'UPDATE'] );

If the role is granted permissions other than those specified, the diagnostics will list the extra permissions, like so:

# Failed test 14: "Role bob should be granted SELECT on columns in widgets" # Extra privileges: # INSERT # UPDATE

Likewise if the role is not granted some of the specified permissions on the table:

# Failed test 15: "Role kurk should be granted SELECT, INSERT, UPDATE on columns in widgets" # Missing privileges: # INSERT # UPDATE

In the event that the test fails because the table in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Role slim should be granted SELECT on columns in widgets" # Table widgets does not exist

If the test fails because the role does not exist, the diagnostics will look something like:

# Failed test 17: "Role slim should be granted SELECT on columns in widgets" # Role slim does not exist

**column\_privs\_are()**

SELECT column\_privs\_are ( :schema, :table, :column, :role, :privileges, :description ); SELECT column\_privs\_are ( :schema, :table, :column, :role, :privileges ); SELECT column\_privs\_are ( :table, :column, :role, :privileges, :description ); SELECT column\_privs\_are ( :table, :column, :role, :privileges );

**Parameters**

:schema

Name of a schema in which to find the table.

:table

Name of a table.

:column

Name of a column.

:role

Name of a user or group role.

:privileges

An array of column privileges the role should be granted to the column.

:description

A short description of the test.

Tests the privileges granted to a role to access a single column. The available column privileges are:

* INSERT
* Referencias
* SELECCIONAR
* UPDATE

Note that column privileges were added in PostgreSQL 8.4, so this function will likely throw an exception on earlier versions.

If the :description argument is omitted, an appropriate description will be created. Ejemplos:

SELECT column\_privs\_are( 'public', 'frobulate', 'id', 'fred', ARRAY['SELECT', 'UPDATE'], 'Fred should be able to select and update frobulate.id' ); SELECT column\_privs\_are( 'widgets', 'name', 'slim', ARRAY['INSERT', 'UPDATE'] );

If the role is granted permissions other than those specified, the diagnostics will list the extra permissions, like so:

# Failed test 14: "Role bob should be granted SELECT on widgets.foo" # Extra privileges: # INSERT # UPDATE

Likewise if the role is not granted some of the specified permissions on the table:

# Failed test 15: "Role kurk should be granted SELECT, INSERT, UPDATE on widgets.foo" # Missing privileges: # INSERT # UPDATE

In the event that the test fails because the table in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Role slim should be granted SELECT on widgets.foo" # Table widgets does not exist

If the test fails because the column does not actually exist or is not visible, the diagnostics will tell you:

# Failed test 17: "Role slim should be granted SELECT on gadgets.foo" # Column gadgets.foo does not exist

Si la prueba falla porque el rol no existe, los diagnósticos se verán como:

# Failed test 18: "Role slim should be granted SELECT on gadgets.foo" # Role slim does not exist

**function\_privs\_are()**

SELECT function\_privs\_are ( :schema, :function, :args, :role, :privileges, :description ); SELECT function\_privs\_are ( :schema, :function, :args, :role, :privileges ); SELECT function\_privs\_are ( :function, :args, :role, :privileges, :description ); SELECT function\_privs\_are ( :function, :args, :role, :privileges );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar la función.

:function

Nombre de una función.

:args

Matriz de argumentos de la función.

:role

Nombre de un usuario o rol de grupo.

:privileges

Una serie de privilegios de función que el rol debe otorgarse a la función.

:description

Una breve descripción de la prueba.

Prueba los privilegios otorgados a un rol para acceder a una función. Los privilegios de función disponibles son:

* EJECUTAR

Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT function\_privs\_are( 'public', 'frobulate', ARRAY['integer'], 'fred', ARRAY['EXECUTE'], 'Fred should be able to execute frobulate(int)' ); SELECT function\_privs\_are( 'bake', '{}', 'slim', '{}');

Si a la función se le otorgan permisos distintos a los especificados, los diagnósticos enumerarán los permisos adicionales, como así:

# Failed test 14: "Role bob should be granted no privileges on foo()" # Extra privileges: # EXECUTE

Del mismo modo, si al rol no se le otorgan algunos de los permisos especificados en la función:

# Failed test 15: "Role kurk should be granted EXECUTE foo()" # Missing privileges: # EXECUTE

En el caso de que la prueba falle porque la función en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "Role slim should be granted EXECUTE on foo(int)" # Function foo(int) does not exist

Si la prueba falla porque el rol no existe, los diagnósticos se verán como:

# Failed test 17: "Role slim should be granted EXECUTE on foo()" # Role slim does not exist

**language\_privs\_are()**

SELECT language\_privs\_are ( :lang, :role, :privileges, :description ); SELECT language\_privs\_are ( :lang, :role, :privileges );

**Parámetros**

:lang

Nombre de un idioma.

:role

Nombre de un usuario o rol de grupo.

:privileges

Una matriz de privilegios de tabla que el rol debe otorgarse al idioma.

:description

Una breve descripción de la prueba.

Prueba los privilegios otorgados a un rol para acceder a un idioma. Los privilegios de función disponibles son:

* USO

Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT language\_privs\_are( 'plpgsql', 'fred', ARRAY['USAGE'], 'Fred should be granted USAGE on language "flipr"' ); SELECT language\_privs\_are( 'plperl', ARRAY['USAGE'] );

Si a la función se le otorgan permisos distintos a los especificados, los diagnósticos enumerarán los permisos adicionales, como así:

# Failed test 14: "Role bob should be granted no privileges on banks" # Extra privileges: # USAGE

Del mismo modo, si a la función no se le otorgan algunos de los permisos especificados en el idioma:

# Failed test 15: "Role kurk should be granted USAGE on banks" # Missing privileges: # USAGE

In the event that the test fails because the language in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Role slim should be granted USAGE on plr" # Language plr does not exist

If the test fails because the role does not exist, the diagnostics will look something like:

# Failed test 17: "Role slim should be granted USAGE on pllolcode" # Role slim does not exist

**fdw\_privs\_are()**

SELECT fdw\_privs\_are ( :fdw, :role, :privileges, :description ); SELECT fdw\_privs\_are ( :fdw, :role, :privileges );

**Parameters**

:fdw

Name of a foreign data wrapper.

:role

Name of a user or group role.

:privileges

An array of table privileges the role should be granted to the foreign data wrapper.

:description

A short description of the test.

Tests the privileges granted to a role to access a foreign data wrapper. The available function privileges are:

* USAGE

Note that foreign data wrapper privileges were added in PostgreSQL 8.4, so this function will likely throw an exception on earlier versions.

If the :description argument is omitted, an appropriate description will be created. Ejemplos:

SELECT fdw\_privs\_are( 'oracle', 'fred', ARRAY['USAGE'], 'Fred should be granted USAGE on fdw "oracle"' ); SELECT fdw\_privs\_are( 'log\_csv', ARRAY['USAGE'] );

If the role is granted permissions other than those specified, the diagnostics will list the extra permissions, like so:

# Failed test 14: "Role bob should be granted no privileges on odbc" # Extra privileges: # USAGE

Likewise if the role is not granted some of the specified permissions on the FDW:

# Failed test 15: "Role kurk should be granted USAGE on odbc" # Missing privileges: # USAGE

In the event that the test fails because the FDW in question does not actually exist or is not visible, you will see an appropriate diagnostic such as:

# Failed test 16: "Role slim should be granted USAGE on FDW sqlite" # FDW sqlite does not exist

If the test fails because the role does not exist, the diagnostics will look something like:

# Failed test 17: "Role slim should be granted USAGE on sqlite" # Role slim does not exist

**server\_privs\_are()**

SELECT server\_privs\_are ( :server, :role, :privileges, :description ); SELECT server\_privs\_are ( :server, :role, :privileges );

**Parameters**

:server

Name of a server.

:role

Name of a user or group role.

:privileges

An array of table privileges the role should be granted to the server.

:description

A short description of the test.

Tests the privileges granted to a role to access a server. The available function privileges are:

* USAGE

Note that server privileges were added in PostgreSQL 8.4, so this function will likely throw an exception on earlier versions.

Si :descriptionse omite el argumento, se creará una descripción apropiada. Ejemplos:

SELECT server\_privs\_are( 'otherdb', 'fred', ARRAY['USAGE'], 'Fred should be granted USAGE on server "otherdb"' ); SELECT server\_privs\_are( 'myserv', ARRAY['USAGE'] );

Si a la función se le otorgan permisos distintos a los especificados, los diagnósticos enumerarán los permisos adicionales, como así:

# Failed test 14: "Role bob should be granted no privileges on myserv" # Extra privileges: # USAGE

Del mismo modo, si a la función no se le otorgan algunos de los permisos especificados en el servidor:

# Failed test 15: "Role kurk should be granted USAGE on oltp" # Missing privileges: # USAGE

En el caso de que la prueba falle porque el servidor en cuestión no existe realmente o no está visible, verá un diagnóstico apropiado como:

# Failed test 16: "Role slim should be granted USAGE on server oltp" # server oltp does not exist

Si la prueba falla porque el rol no existe, los diagnósticos se verán como:

# Failed test 17: "Role slim should be granted USAGE on oltp" # Role slim does not exist

**policies\_are()**

SELECT policies\_are( :schema, :table, :policies, :description ); SELECT policies\_are( :schema, :table, :policies ); SELECT policies\_are( :table, :policies, :description ); SELECT policies\_are( :table, :policies );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :table.

:table

Nombre de una tabla en la que encontrar las políticas.

:policies

Una matriz de nombres de políticas.

:description

Una breve descripción de la prueba.

Esta función prueba que todas las políticas en la tabla nombrada son solo las políticas que *deberían* estar en esa tabla. Si :schemase omite el argumento, la tabla debe estar visible en la ruta de búsqueda, excluyendo pg\_catalogy information\_schema. Si se omite la descripción, se generará una descripción predeterminada generalmente útil. Ejemplo:

SELECT policies\_are( 'myschema', 'atable', ARRAY[ 'atable\_policy\_one', 'atable\_policy\_two' ] );

En el caso de una falla, verá los diagnósticos que enumeran las políticas adicionales y / o faltantes, como por ejemplo:

# Failed test 13: "Table myschema.atable should have the correct policies" # Extra policies: # policy\_for\_atable # Missing policies: # atable\_policy\_two

**policy\_roles\_are()**

SELECT policy\_roles\_are( :schema, :table, :policy, :roles, :description ); SELECT policy\_roles\_are( :schema, :table, :policy, :roles ); SELECT policy\_roles\_are( :table, :policy, :roles, :description ); SELECT policy\_roles\_are( :table, :policy, :roles );

**Parámetros**

:schema

Nombre de un esquema en el que encontrar el :table.

:table

Nombre de una tabla a la que se :policyaplica.

:policy

Nombre de una póliza.

:roles

Una matriz de nombres de rol a la que :policyse aplica.

:description

Una breve descripción de la prueba.

This function tests whether the roles to which policy applies are only the roles that *should* be on that policy. If the :schema argument is omitted, the table must be visible in the search path, excluding pg\_catalog and information\_schema . If the description is omitted, a generally useful default description will be generated. Ejemplo:

SELECT policy\_roles\_are( 'myschema', 'atable', 'apolicy' ARRAY[ 'atable\_apolicy\_role\_one', 'atable\_apolicy\_role\_two' ] );

In the event of a failure, you'll see diagnostics listing the extra and/or missing policy roles, like so:

# Failed test 13: "Policy apolicy for table myschema.atable should have the correct roles" # Extra policy roles: # arole\_one # Missing policy roles: # atable\_apolicy\_role\_one

**policy\_cmd\_is()**

SELECT policy\_cmd\_is( :schema, :table, :policy, :command, :description ); SELECT policy\_cmd\_is( :schema, :table, :policy, :command ); SELECT policy\_cmd\_is( :table, :policy, :command, :description ); SELECT policy\_cmd\_is( :table, :policy, :command );

**Parameters**

:schema

Name of a schema in which to find the :table .

:table

Name of a table to which :policy applies.

:policy

Name of a policy.

:command

The command type to which the :policy is applied.

:description

A short description of the test.

This function tests whether the command to which policy applies is same as command that is given in function arguments.

Los :commandtipos de pólizas disponibles son:

* SELECCIONAR
* INSERTAR
* ACTUALIZAR
* BORRAR
* TODOS

Si :schemase omite el argumento, la tabla debe estar visible en la ruta de búsqueda, excluyendo pg\_catalogy information\_schema. Si :descriptionse omite (pero :schemaestá presente, asegúrese de convertir el :policynombre), se generará una descripción predeterminada generalmente útil. Ejemplo:

SELECT policy\_cmd\_is( 'myschema', 'atable', 'apolicy'::NAME 'all' );

En el caso de una falla, verá los diagnósticos que listan el comando de política adicional y / o faltante, así:

# Failed test 13: "Policy apolicy for table myschema.atable should apply to ALL command" # have: INSERT # want: ALL

**Ninguna prueba para los malvados**

Hay más para pgTAP. Oh, *así que* mucho más! Puede dar salida a sus propios [diagnósticos](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Diagnostics) . Puede escribir [pruebas condicionales](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Conditional+Tests) basadas en la salida de las [funciones](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Utility+Functions) de [utilidad](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Utility+Functions) . Puedes hacer [lotes de pruebas en funciones](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg#Tap+That+Batch) . Sigue leyendo para aprender todo sobre esto.

**Diagnósticos**

Si elige la función de prueba correcta, normalmente obtendrá una buena idea de lo que salió mal cuando falló. Pero a veces no funciona de esa manera. Así que aquí tenemos formas para que escriba sus propios mensajes de diagnóstico que son más seguros que simplemente \echoo SELECT foo.

**diag()**

SELECT diag( :lines );

**Parámetros**

:lines

Una lista de uno o más valores de SQL del mismo tipo.

Devuelve un mensaje de diagnóstico que garantiza que no interfiere con la salida de prueba. Práctico para este tipo de cosas:

-- Output a diagnostic message if the collation is not en\_US.UTF-8. SELECT diag( E'These tests expect LC\_COLLATE to be en\_US.UTF-8,\n', 'but yours is set to ', setting, E'.\n', 'As a result, some tests may fail. YMMV.' ) FROM pg\_settings WHERE name = 'lc\_collate' AND setting <> 'en\_US.UTF-8';

Lo que produciría:

# These tests expect LC\_COLLATE to be en\_US.UTF-8, # but yours is set to en\_US.ISO8859-1. # As a result, some tests may fail. YMMV.

Puede pasar datos de cualquier tipo a diag()PostgreSQL 8.3 y superior y todos se convertirán en texto para los diagnósticos. En PostgreSQL 8.4 y superior, puede pasar cualquier número de argumentos (siempre que sean todos del mismo tipo de datos) y se concatenarán juntos.

**Pruebas condicionales**

A veces, ejecutar una prueba bajo ciertas condiciones hará que el script o la función de la prueba muera. Una determinada función o función no está implementada (como sha256()antes de PostgreSQL 11), algunos recursos no están disponibles (como un lenguaje de procedimiento) o un módulo contrib no está disponible. En estos casos, es necesario omitir las pruebas o declarar que se supone que deben fallar pero que funcionarán en el futuro (una prueba de todo).

**skip()**

SELECT skip( :why, :how\_many ); SELECT skip( :how\_many, :why ); SELECT skip( :why ); SELECT skip( :how\_many );

**Parámetros**

:why

Razón para saltarse las pruebas.

:how\_many

Número de pruebas a saltar

Resultados de prueba SKIP salidas. Úselo en una expresión condicional dentro de una SELECTdeclaración para reemplazar la salida de una prueba que de otro modo habría ejecutado.

SELECT CASE WHEN pg\_version\_num() < 80300 THEN skip('has\_enum() not supported before 8.3', 2 ) ELSE collect\_tap( has\_enum( 'bug\_status' ), has\_enum( 'bug\_status', 'mydesc' ) ) END;

Observe cómo CASEse ha usado el uso de la declaración condicional para determinar si se deben ejecutar un par de pruebas. Si se van a ejecutar, se ejecutan collect\_tap()para que podamos ejecutar algunas pruebas en la misma consulta. Si no queremos ejecutarlos, llamamos skip()y le decimos cuántas pruebas nos estamos saltando.

Si no especifica cuántas pruebas omitir, skip()asumirá que está omitiendo solo una. Esto es útil para el caso simple, por supuesto:

SELECT CASE current\_schema() WHEN 'public' THEN is( :this, :that ) ELSE skip( 'Tests not running in the "public" schema' ) END;

Pero también puede usarlo en una SELECTdeclaración que de lo contrario devolvería varias filas:

SELECT CASE current\_schema() WHEN 'public' THEN is( nspname, 'public' ) ELSE skip( 'Cannot see the public schema' ) END FROM pg\_namespace;

Esto hará que se omita el mismo número de filas que se hubiera probado si la WHENcondición fuera cierta.

**todo()**

SELECT todo( :why, :how\_many ); SELECT todo( :how\_many, :why ); SELECT todo( :why ); SELECT todo( :how\_many );

**Parámetros**

:why

Motivo de las pruebas de marcado en cuanto a dos.

:how\_many

Número de pruebas a marcar en cuanto a dos.

Declares a series of tests that you expect to fail and why. Perhaps it's because you haven't fixed a bug or haven't finished a new feature:

SELECT todo('URIGeller not finished', 2); \set card '\'Eight of clubs\'' SELECT is( URIGeller.yourCard(), :card, 'Is THIS your card?' ); SELECT is( URIGeller.bendSpoon(), 'bent', 'Spoon bending, how original' );

With todo() , :how\_many specifies how many tests are expected to fail. If :how\_many is omitted, it defaults to 1. pgTAP will run the tests normally, but print out special flags indicating they are “todo” tests. The test harness will interpret these failures as ok. Should any todo test pass, the harness will report it as an unexpected success. You then know that the thing you had todo is done and can remove the call to todo() .

The nice part about todo tests, as opposed to simply commenting out a block of tests, is that they're like a programmatic todo list. You know how much work is left to be done, you're aware of what bugs there are, and you'll know immediately when they're fixed.

**todo\_start( why )**

**todo\_start( )**

This function allows you declare all subsequent tests as TODO tests, up until the todo\_end() function is called.

The todo() syntax is generally pretty good about figuring out whether or not we're in a TODO test. However, often we find it difficult to specify the *number* of tests that are TODO tests. Thus, you can instead use todo\_start() and todo\_end() to more easily define the scope of your TODO tests.

Note that you can nest TODO tests, too:

SELECT todo\_start('working on this'); -- lots of code SELECT todo\_start('working on that'); -- more code SELECT todo\_end(); SELECT todo\_end();

This is generally not recommended, but large testing systems often have weird internal needs.

The todo\_start() and todo\_end() function should also work with the todo() function, although it's not guaranteed and its use is also discouraged:

SELECT todo\_start('working on this'); -- lots of code SELECT todo('working on that', 2); -- Two tests for which the above line applies -- Followed by more tests scoped till the following line. SELECT todo\_end();

We recommend that you pick one style or another of TODO to be on the safe side.

**todo\_end()**

Stops running tests as TODO tests. This function is fatal if called without a preceding todo\_start() method call.

**in\_todo()**

Returns true if the test is currently inside a TODO block.

**Utility Functions**

Along with the usual array of testing, planning, and diagnostic functions, pTAP provides a few extra functions to make the work of testing more pleasant.

**pgtap\_version()**

SELECT pgtap\_version();

Returns the version of pgTAP installed in the server. The value is NUMERIC , and thus suitable for comparing to a decimal value:

SELECT CASE WHEN pgtap\_version() < 0.17 THEN skip('No sequence assertions before pgTAP 0.17') ELSE has\_sequence('my\_big\_seq') END;

**pg\_version()**

SELECT pg\_version();

Returns the server version number against which pgTAP was compiled. This is the stringified version number displayed in the first part of the core version() function and stored in the “server\_version” setting:

try=% select current\_setting( 'server\_version'), pg\_version(); current\_setting | pg\_version -----------------+------------ 8.3.4 | 8.3.4 (1 row)

**pg\_version\_num()**

SELECT pg\_version\_num();

Returns an integer representation of the server version number against which pgTAP was compiled. This function is useful for determining whether or not certain tests should be run or skipped (using skip() ) depending on the version of PostgreSQL. Por ejemplo:

SELECT CASE WHEN pg\_version\_num() < 80300 THEN skip('has\_enum() not supported before 8.3' ) ELSE has\_enum( 'bug\_status', 'mydesc' ) END;

The revision level is in the tens position, the minor version in the thousands position, and the major version in the ten thousands position and above (assuming PostgreSQL 10 is ever released, it will be in the hundred thousands position). This value is the same as the server\_version\_num setting.

**os\_name()**

SELECT os\_name();

Returns a string representing the name of the operating system on which pgTAP was compiled. This can be useful for determining whether or not to skip tests on certain operating systems.

This is usually the same a the output of uname , but converted to lower case. There are some semantics in the pgTAP build process to detect other operating systems, though assistance in improving such detection would be greatly appreciated.

**NOTE:** The values returned by this function may change in the future, depending on how good the pgTAP build process gets at detecting a OS.

**collect\_tap()**

SELECT collect\_tap(:lines);

**Parameters**

:lines

A list of one or more lines of TAP.

Collects the results of one or more pgTAP tests and returns them all. Useful when used in combination with skip() :

SELECT CASE os\_name() WHEN 'darwin' THEN collect\_tap( cmp\_ok( 'Bjørn'::text, '>', 'Bjorn', 'ø > o' ), cmp\_ok( 'Pınar'::text, '>', 'Pinar', 'ı > i' ), cmp\_ok( 'José'::text, '>', 'Jose', 'é > e' ), cmp\_ok( 'Täp'::text, '>', 'Tap', 'ä > a' ) ) ELSE skip('Collation-specific test', 4) END;

On PostgreSQL 8.4 and higher, it can take any number of arguments. Lower than 8.4 requires the explicit use of an array:

SELECT collect\_tap(ARRAY[ ok(true, 'This should pass'), ok(false, 'This should fail) ]);

**display\_oper()**

SELECT display\_oper( :opername, :operoid );

**Parameters**

:opername

Operator name.

:operoid

Operator OID.

Similar to casting an operator OID to regoperator , only the schema is not included in the display. Por ejemplo:

SELECT display\_oper(oprname, oid ) FROM pg\_operator;

Used internally by pgTAP to compare operators, but may be more generally useful.

**pg\_typeof()**

SELECT pg\_typeof(:any);

**Parameters**

:any

Any SQL value.

Returns a regtype identifying the type of value passed to the function. This function is used internally by cmp\_ok() to properly construct types when executing the comparison, but might be generally useful.

try=% select pg\_typeof(12), pg\_typeof(100.2); pg\_typeof | pg\_typeof -----------+----------- integer | numeric

*Note:* pgTAP does not build pg\_typeof() on PostgreSQL 8.4 or higher, because it's in core in 8.4. You only need to worry about this if you depend on the function being in particular schema. It will always be in pg\_catalog in 8.4 and higher.

**findfuncs()**

SELECT findfuncs( :schema, :pattern, :exclude\_pattern ); SELECT findfuncs( :schema, :pattern ); SELECT findfuncs( :pattern, :exclude\_pattern ); SELECT findfuncs( :pattern );

**Parameters**

:schema

Schema to search for functions.

:pattern

Regular expression pattern against which to match function names.

:pattern

Regular expression pattern to exclude functions with matching names.

This function searches the named schema or, if no schema is passed, the search patch, for all functions that match the regular expression pattern. The optional exclude regular expression pattern can be used to prevent matchin startup/setup/teardown/shutdown functions.

The functions it finds are returned as an array of text values, with each value consisting of the schema name, a dot, and the function name. Por ejemplo:

SELECT findfuncs('tests', '^test); findfuncs ----------------------------------- {tests.test\_foo,tests."test bar"} (1 row)

**Toque ese lote**

A veces puede ser útil agrupar muchas pruebas TAP en una función. La forma más sencilla de hacerlo es definir una función que, a RETURNS SETOF TEXTcontinuación, simplemente llamar RETURN NEXTpara cada prueba TAP. Aquí hay un ejemplo simple:

CREATE OR REPLACE FUNCTION my\_tests( ) RETURNS SETOF TEXT AS $$ BEGIN RETURN NEXT pass( 'plpgsql simple' ); RETURN NEXT pass( 'plpgsql simple 2' ); END; $$ LANGUAGE plpgsql;

Luego, simplemente puede llamar a la función para ejecutar todas sus pruebas TAP a la vez:

SELECT plan(2); SELECT \* FROM my\_tests(); SELECT \* FROM finish();

**do\_tap()**

SELECT do\_tap( :schema, :pattern ); SELECT do\_tap( :schema ); SELECT do\_tap( :pattern ); SELECT do\_tap();

**Parámetros**

:schema

Nombre de un esquema que contiene funciones de prueba pgTAP.

:pattern

Patrón de expresión regular contra el cual hacer coincidir los nombres de las funciones.

Si lo desea, puede crear una gran cantidad de estas funciones de toque por lotes y luego usar la do\_tap()función para ejecutarlas todas a la vez. Si no se pasa ningún argumento, intentará encontrar todas las funciones visibles que comiencen con "prueba". Si se pasa un nombre de esquema, buscará y ejecutará funciones de prueba solo en ese esquema (asegúrese de convertir el esquema namesi es el único argumento). Si se pasa un patrón de expresión regular, buscará nombres de funciones que coincidan con ese patrón en la ruta de búsqueda. Si se aprueban ambos, por supuesto, solo buscará funciones de prueba que coincidan con la función en el esquema nombrado.

Esto puede ser muy útil si prefiere mantener todas sus pruebas TAP en funciones definidas en la base de datos. Simplemente llame plan(), use do\_tap()para ejecutar todas sus pruebas, y luego llame finish(). Un simple ejemplo muerto:

SELECT plan(32); SELECT \* FROM do\_tap('testschema'::name); SELECT \* FROM finish();

Como beneficio adicional, si client\_min\_messagesse establece en "advertencia", "error", "fatal" o "pánico", el nombre de cada función se emitirá como un mensaje de diagnóstico antes de que se llame. Por ejemplo, si se do\_tap()encuentran y ejecutan dos funciones de prueba de TAP y client\_min\_messagesse establece una "advertencia", la salida tendrá un aspecto similar al siguiente:

# public.test\_this() ok 1 - simple pass ok 2 - another simple pass # public.test\_that() ok 3 - that simple ok 4 - that simple 2

Lo que hará que sea mucho más fácil decir qué funciones deben ser examinadas para las pruebas que fallan.

**runtests()**

SELECT runtests( :schema, :pattern ); SELECT runtests( :schema ); SELECT runtests( :pattern ); SELECT runtests( );

**Parámetros**

:schema

Nombre de un esquema que contiene funciones de prueba pgTAP.

:pattern

Patrón de expresión regular contra el cual hacer coincidir los nombres de las funciones.

If you'd like pgTAP to plan, run all of your test functions, and finish, all in one fell swoop, use runtests() . This most closely emulates the xUnit testing environment, similar to the functionality of [PGUnit](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://en.dklab.ru/lib/dklab_pgunit/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhglXXvDkRMYnMXlpeIthrLEjj4KWg) . Ejemplo:

SELECT \* FROM runtests( 'testschema', '^test' );

As with do\_tap() , you can pass in a schema argument and/or a pattern that the names of the tests functions can match. If you pass in only the schema argument, be sure to cast it to name to identify it as a schema name rather than a pattern:

SELECT \* FROM runtests('testschema'::name);

Unlike do\_tap() , runtests() fully supports startup, shutdown, setup, and teardown functions, as well as transactional rollbacks between tests. It also outputs the test plan, executes each test function as a TAP subtest, and finishes the tests, so you don't have to call plan() or finish() yourself. The output, assuming a single startup test, two subtests, and a single shutdown test, will look something like this:

ok 1 - Startup test # Subtest: public.test\_this() ok 1 - simple pass ok 2 - another simple pass ok 2 - public.test\_this() # Subtest: public.test\_that() ok 1 - that simple ok 2 - that simple 2 ok 3 - public.test\_that() ok 4 - Shutdown test 1..4

The fixture functions run by runtests() are as follows:

* ^startup - Functions whose names start with “startup” are run in alphabetical order before any test functions are run.
* ^setup - Functions whose names start with “setup” are run in alphabetical order before each test function is run.
* ^teardown - Functions whose names start with “teardown” are run in alphabetical order after each test function is run. They will not be run, however, after a test that has died.
* ^shutdown - Functions whose names start with “shutdown” are run in alphabetical order after all test functions have been run.

Note that all tests executed by runtests() are run within a single transaction, and each test is run in a subtransaction that also includes execution all the setup and teardown functions. All transactions are rolled back after each test function, and at the end of testing, leaving your database in largely the same condition as it was in when you started it (the one exception I'm aware of being sequences, which are not rolled back to the value used at the beginning of a rolled-back transaction).

**Secrets of the pgTAP Mavens**

Over the years, a number of techniques have evolved to make all of our pgTAP testing lives easier. Here are some of them.

**Relational-style Loops**

Need to test a bunch of objects and find yourself looking for some kind of for loop to [DRY](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://en.wikipedia.org/wiki/Don%2527t_repeat_yourself&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjfiG2J2bgWp82kmuzh6v21-jS5nw) off with? SQL doesn't have one, of course, but that's because it doesn't need one: the whole language is built around doing things to a bunch of rows. So take advantage of it: build relations with the [VALUES](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://www.postgresql.org/docs/current/static/sql-values.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhhFQ_B52TYu37AZIEZ3r2s4LBry_g) command! For example, to make sure you have a table in a defined list of schemas, try something like this:

SELECT has\_table(sch, 'widgets', format('Has %I.widgets', sch)) FROM (VALUES('amazon'), ('starbucks'), ('boeing')) F(sch);

Note the use of the [format function](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://www.postgresql.org/docs/current/static/functions-string.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhibZRp7nU42sPDhVv--tdUhPs4hWQ#functions-string-format) to make a nice test description, too. Here's a more complicated example that uses a cross join to test that various columns are NOT NULL in a specific table in a bunch of schemas:

SELECT col\_not\_null(sch, 'table1', col) FROM (VALUES('schema1'), ('schema1')) AS stmp (sch) CROSS JOIN (VALUES('col\_pk'), ('col2'), ('col3')) AS ctmp (col);

**Compose Yourself**

So, you've been using pgTAP for a while, and now you want to write your own test functions. Go ahead; I don't mind. In fact, I encourage it. ¿Cómo? Why, by providing a function you can use to test your tests, of course!

But first, a brief primer on writing your own test functions. There isn't much to it, really. Just write your function to do whatever comparison you want. As long as you have a boolean value indicating whether or not the test passed, you're golden. Just then use ok() to ensure that everything is tracked appropriately by a test script.

For example, say that you wanted to create a function to ensure that two text values always compare case-insensitively. Sure you could do this with is() and the LOWER() function, but if you're doing this all the time, you might want to simplify things. Here's how to go about it:

CREATE OR REPLACE FUNCTION lc\_is (text, text, text) RETURNS TEXT AS $$ DECLARE result BOOLEAN; BEGIN result := LOWER($1) = LOWER($2); RETURN ok( result, $3 ) || CASE WHEN result THEN '' ELSE E'\n' || diag( ' Have: ' || $1 || E'\n Want: ' || $2; ) END; END; $$ LANGUAGE plpgsql;

Yep, that's it. The key is to always use pgTAP's ok() function to guarantee that the output is properly formatted, uses the next number in the sequence, and the results are properly recorded in the database for summarization at the end of the test script. You can also provide diagnostics as appropriate; just append them to the output of ok() as we've done here.

Por supuesto, no tienes que usar directamente ok(); También puede usar otra función pgTAP que finalmente llama ok(). IOW, mientras que el ejemplo anterior es instructivo, esta versión es más fácil a la vista:

CREATE OR REPLACE FUNCTION lc\_is ( TEXT, TEXT, TEXT ) RETURNS TEXT AS $$ SELECT is( LOWER($1), LOWER($2), $3); $$ LANGUAGE sql;

Pero de cualquier manera, deje que pgTAP maneje los resultados de la prueba y el formato de la salida.

**Prueba de funciones de prueba**

Ahora has escrito tu función de prueba. Entonces, ¿cómo lo pruebas? ¡Por qué, con esta práctica función de prueba!

**check\_test()**

SELECT check\_test( :test\_output, :is\_ok, :name, :want\_description, :want\_diag, :match\_diag ); SELECT check\_test( :test\_output, :is\_ok, :name, :want\_description, :want\_diag ); SELECT check\_test( :test\_output, :is\_ok, :name, :want\_description ); SELECT check\_test( :test\_output, :is\_ok, :name ); SELECT check\_test( :test\_output, :is\_ok );

**Parámetros**

:schema

Nombre de un esquema que contiene funciones de prueba pgTAP.

:test\_output

La salida de su prueba. Por lo general, solo se devuelve mediante una llamada a la función de prueba en sí. Necesario.

:is\_ok

Booleano que indica si se espera que la prueba pase o no. Necesario.

:name

Un breve nombre para su prueba, para que sea más fácil encontrar fallas en su script de prueba. Opcional.

:want\_description

Se espera que la descripción de la prueba sea emitida por la prueba. Opcional. Utilice una cadena vacía para comprobar que no se genera ninguna descripción.

:want\_diag

Salida de mensaje de diagnóstico esperado durante la ejecución de una prueba. Siempre debe seguir lo que salga de la llamada a ok(). Opcional. Utilice una cadena vacía para comprobar que no se genera ninguna descripción.

:match\_diag

Se usa matches()para comparar los diagnósticos en lugar de :is(). Es útil para aquellas situaciones en las que no está seguro de lo que habrá en la salida, pero puede coincidir con una expresión regular.

Esta función se ejecuta en cualquier lugar entre una y tres pruebas contra una función de prueba. En su forma más simple, simplemente pasa la salida de su función de prueba (y debe ser una y **solo una** salida de la función de prueba, o arruinará la cuenta, así que no haga eso) y un valor booleano que indique si o no esperas que la prueba haya pasado. Eso se ve algo como esto:

SELECT \* FROM check\_test( lc\_eq('This', 'THIS', 'eq'), true );

Todos los demás argumentos son opcionales, pero le recomiendo que *siempre* incluya un nombre corto de prueba para que sea más fácil localizar las fallas en su script de prueba. check\_test()utiliza este nombre para construir descripciones de todas las pruebas que ejecuta. Por ejemplo, sin un nombre corto, el ejemplo anterior producirá una salida como la siguiente:

not ok 14 - Test should pass

Sí, pero que prueba? Así que dale un nombre muy breve y sabrás qué prueba. Si tienes muchos de estos, no será de mucha ayuda. Así que dale a cada llamada check\_test()un nombre:

SELECT \* FROM check\_test( lc\_eq('This', 'THIS', 'eq'), true, 'Simple lc\_eq test', );

Entonces obtendrás una salida más como esta:

not ok 14 - Simple lc\_test should pass

Lo que hará que sea mucho más fácil encontrar la prueba que falla en su script de prueba.

El cuarto argumento opcional es la descripción que espera obtener. Esto es especialmente importante si su función de prueba genera una descripción cuando no se le pasa ninguna. ¡Quiere asegurarse de que su función genere la descripción de la prueba que cree que debería! Esto hará que se ejecute una segunda prueba en la función de prueba. Así que para algo como esto:

SELECT \* FROM check\_test( lc\_eq( ''this'', ''THIS'' ), true, 'lc\_eq() test', 'this is THIS' );

La salida se vería así, asumiendo que la lc\_eq()función generó la descripción correcta (el ejemplo anterior no lo hace):

ok 42 - lc\_eq() test should pass ok 43 - lc\_eq() test should have the proper description

¿Ves cómo se ejecutan dos pruebas para una sola llamada a check\_test()? Asegúrese de ajustar su plan en consecuencia. También tenga en cuenta cómo se usó el nombre de la prueba en las descripciones de ambas pruebas.

Si la prueba hubiera fallado, daría un buen diagnóstico. Internamente solo usa is()para comparar las cuerdas:

# Failed test 43: "lc\_eq() test should have the proper description" # have: 'this is this' # want: 'this is THIS'

El quinto argumento, :want\_diagque también es opcional, compara los diagnósticos generados durante la prueba con una cadena esperada. Dichos diagnósticos **deben** seguir lo que salga de la llamada ok()en su prueba. Su función de prueba no debe llamar diag()hasta después de que llame ok()o las cosas se pondrán realmente funky.

Suponiendo que ha seguido esa regla en su lc\_eq()función de prueba, vea qué sucede cuando lc\_eq()falla una . Escribe tu prueba para probar los diagnósticos así:

SELECT \* FROM check\_test( lc\_eq( ''this'', ''THat'' ), false, 'lc\_eq() failing test', 'this is THat', E' Want: this\n Have: THat );

Esto, por supuesto, activa una tercera prueba para ejecutar. La salida se verá así:

ok 44 - lc\_eq() failing test should fail ok 45 - lc\_eq() failing test should have the proper description ok 46 - lc\_eq() failing test should have the proper diagnostics

Y, por supuesto, si la prueba de diagnóstico falla, generará diagnósticos como lo haría una falla de descripción, algo como esto:

# Failed test 46: "lc\_eq() failing test should have the proper diagnostics" # have: Have: this # Want: that # want: Have: this # Want: THat

Si pasa el sexto argumento opcional :match\_diag, el :want\_diagargumento se comparará con la salida de diagnóstico real utilizando en matches()lugar de is(). Esto le permite usar una expresión regular en el :want\_diagargumento para que coincida con la salida, para aquellas situaciones en las que alguna parte de la salida puede variar, como los diagnósticos basados ​​en el tiempo.

Me doy cuenta de que todo esto puede ser un poco confuso, dados los diversos deseos y necesidades, pero hace el trabajo. Por supuesto, si sus diagnósticos utilizan algo que no sea sangrado "tener" y "querer", tales fallas serán más fáciles de leer. Pero de cualquier manera, *¡* prueba tus diagnósticos!

**Compatibilidad**

Aquí hay algunas notas sobre cómo se construye pgTAP para versiones particulares de PostgreSQL. Esto te ayuda a entender cualquier efecto secundario. Si prefiere no tener estos cambios en su esquema, cree pgTAPun esquema solo para ello:

make TAPSCHEMA=tap

Para ver los detalles de cada versión de PostgreSQL, consulte los archivos en el compat/directorio en la distribución pgTAP.

**10 y hasta**

Sin cambios. Todo debería funcionar.

**9.6 y abajo**

* Las funciones de prueba de partición no están disponibles, porque las particiones no se introdujeron hasta 10.

**9.4 y abajo**

* lives\_ok () y throws\_ok () no interceptarán ASSERT\_FAILURE, ya que las afirmaciones no existen antes de 9.5.

**9.2 y abajo**

* La salida de diagnóstico lives\_ok()y las excepciones de la función xUnit no incluirán información de esquema, tabla, columna, tipo de datos o restricción, ya que dichos diagnósticos no se introdujeron hasta 9.3.

**9.1 y abajo**

* Las lives\_ok()excepciones de salida de diagnóstico y xUnit no generarán un error en el contexto o los detalles, ya que dichos diagnósticos no se introdujeron hasta 9.2.

**9.0 y abajo**

* La foreign\_table\_owner\_is()función no funcionará, porque, por supuesto, no hubo tablas externas hasta 9.1.
* Las extensions\_are()funciones no están disponibles, porque las extensiones no se introdujeron hasta 9.1.

**8.4 y abajo**

* La sequence\_privs\_are()función no funciona, porque no se pudieron otorgar privilegios en secuencias anteriores a 9.0.
* La triggers\_are()función no ignora los disparadores creados internamente.

**8.3 y abajo**

* Se aplica un parche para modificar results\_eq()y row\_eq()convertir registros a texto antes de compararlos. Esto significa que las cosas serán principalmente correctas, pero también significa que dos consultas con tipos incompatibles que se convierten a la misma cadena de texto pueden considerarse incorrectamente equivalentes.
* La función AC,, pg\_typeof()está construida e instalada en un DSO. Esto es para compatibilidad con la misma función que se envía en el núcleo 8.4, y se requiere para cmp\_ok()y isa\_ok()para funcionar.
* Las formas variad diag()y collect\_tap()no están disponibles. collect\_tap()Sin embargo, puede pasar una matriz de TAP a .
* Estas funciones de prueba de permisos no funcionan, porque no se pueden otorgar permisos sobre los objetos relevantes hasta 8.4:
  + has\_any\_column\_privilege()
  + has\_column\_privilege()
  + has\_foreign\_data\_wrapper\_privilege()
  + has\_server\_privilege()
* Estas funciones de prueba de herencia no están disponibles, ya que internamente utilizan una consulta de expresión de tabla común recursiva que no se admite antes de 8.4:
  + is\_ancestor\_of()
  + isnt\_ancestor\_of()
  + is\_descendent\_of()
  + isnt\_descendent\_of()

**Metadatos**

**Repositorio publico**

El código fuente de pgTAP está disponible en [GitHub](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://github.com/theory/pgtap/tree/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjRodvlqAF54X-tWdC0GSy0KA1h9w) . Por favor, siéntase libre de tenedor y contribuir!

**Lista de correo**

Únase a la comunidad pgTAP suscribiéndose a la [lista de correo de pgtap-users](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://groups.google.com/forum/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhgizWnjtYLSkPlslBdXLgZAXD661g#!forum/pgtap-users) . Todas las preguntas, comentarios, sugerencias e informes de errores son bienvenidos allí.

**Autor**

[David E. Wheeler](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://justatheory.com/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhiTmcRGbeGo_Yai-V42VM9eWeddHw)

**Creditos**

* Michael Schwern y cromático para Test :: Más.
* Adrian Howard para Test :: Exception.

**Copyright y licencia**

Derechos de autor (c) 2008–2018 David E. Wheeler. Algunos derechos reservados.

Se concede permiso para usar, copiar, modificar y distribuir este software y su documentación para cualquier propósito, sin cargo y sin un acuerdo por escrito, siempre que el aviso de copyright anterior y este párrafo y los dos párrafos siguientes aparezcan en todas las copias.

EN NINGÚN CASO, DAVID E. WHEELER SERÁ RESPONSABLE DE CUALQUIER PARTE POR DAÑOS DIRECTOS, INDIRECTOS, ESPECIALES, INCIDENTALES O CONSECUENTES, INCLUIDOS LOS BENEFICIOS PERDIDOS, RESULTANTES DEL USO DE ESTE SOFTWARE Y SU DOCUMENTACIÓN, INCLUSO SI DAVID E. EEELER TIENE TODO LO QUE SEA. AVISO DE LA POSIBILIDAD DE DICHOS DAÑOS.

DAVID E. WHEELER RECHAZA ESPECÍFICAMENTE CUALQUIER GARANTÍA, INCLUYENDO, PERO NO LIMITADO A, LAS GARANTÍAS IMPLÍCITAS DE COMERCIABILIDAD Y ADECUACIÓN PARA UN PROPÓSITO PARTICULAR. EL SOFTWARE QUE SE PROPORCIONA AQUÍ ESTÁ EN UNA BASE "TAL CUAL", Y DAVID E. WHEELER NO TIENE OBLIGACIONES DE PROPORCIONAR MANTENIMIENTO, SOPORTE, ACTUALIZACIONES, MEJORAS O MODIFICACIONES.

* [Casa](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhh7Q6U1xMScdCwqk24wuMLiwvHOrw)
* [Descargar](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://pgxn.org/dist/pgtap/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhhe-kjoaWkWsafYcXSfAGI6RHlzTg)
* [Documentación](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/documentation.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjtaJVetI2Sz0rDn5MmqHYam_iGdg)
* [pg\_prove](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/pg_prove.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhi1DhMC3IgJGJ6x_AGahjpzOIzqiw)
* [Integración](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://pgtap.org/integration.html&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjiay-SFFY6KEPHHNSrMN0TJ185KQ)
* [Lista de correo](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://groups.google.com/forum/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhgizWnjtYLSkPlslBdXLgZAXD661g#!forum/pgtap-users)
* [GitHub](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=https://github.com/theory/pgtap/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhjJq2kwxJLJwPR2nET4OzEcSIYIzQ)
* [Código: David E. Wheeler](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://theory.pm/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhi2Cyq5aCE3xVBgPRVVXqhaofV-2g)
* [Diseño web: tri-star](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://www.tristarwebdesign.co.uk/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhhihQpOH7FM1IdcrFQOiQ70jZErgg)
* [De archivo: Courtland Whited](https://translate.googleusercontent.com/translate_c?depth=1&hl=es&prev=search&rurl=translate.google.com&sl=en&sp=nmt4&u=http://flickr.com/photos/idreaminir/&xid=17259,15700022,15700186,15700191,15700253,15700256,15700259&usg=ALkJrhgam1DuPr2zYlpXIJMp0W6En0oySw)