İSİM

CREATE TYPE – yeni bir veri türü tanımlar

KULLANIM

```
CREATE TYPE isim AS
    ( öznitelik_ismi veri_türü [, ... ] )

CREATE TYPE isim (
    INPUT = girdi_işlevi,
    OUTPUT = çıktı_işlevi
    [ , RECEIVE = alış_işlevi ]
    [ , SEND = gönderim_işlevi ]
    [ , ANALYZE = analiz_işlevi ]
    [ , INTERNALLENGTH = { dahili_uzunluk | VARIABLE } ]
    [ , PASSEDBYVALUE ]
    [ , ALIGNMENT = hizalama ]
    [ , STORAGE = strateji ]
    [ , DEFAULT = öntanımlı ]
    [ , ELEMENT = öğe_türü ]
    [ , DELIMITER = ayraç ]
)
```

Açıklama

CREATE TYPE o anki veritabanında kullanmak için yeni bir veri türü tanımlar. Türü tanımlayan kullanıcı türün sahibi haline gelir.

Eğer bir şema ismi belirtilmişse, tür bu şemada oluşturulur. Aksi takdirde, tür o anki şemada oluşturulur. Tür ismi aynı şema içinde mevcut veri türü ve veri alanı isimlerinden farklı olmalıdır. (Tablolar ilişkili veri türlerine sahip olduğundan, tür ismi ayrıca, aynı şemada mevcut tablo isimlerinden de farklı olmalıdır.)

Bileşik Türler

CREATE TYPE'ın ilk biçimi bir bileşik veri türü oluşturur. Bileşik tür, öznitelik isimleri ile veri türlerinin bir listesi olarak belirtilir. Bu aslında bir tablonun satır türü ile aynıdır, fakat bir tür tanımlanmak istendiğinde **CREATE TYPE** kullanarak gerçek bir tablo oluşturma ihtiyacı önlenir. Tek başına bir bileşik tür bir işlevin argümanı veya dönüş türü olarak yararlıdır.

Temel Türler

CREATE TYPE'ın ikinci biçimi yeni bir temel tür oluşturur. Parametrelerin yukarıdaki sırayla belirtilmeleri şart değildir, herhangi bir sırayla belirtilebilirler, ayrıca çoğu isteğe bağlıdır. Bir türü tanımlamadan önce **CREATE FUNCTION** kullanarak iki veya daha fazla işlev tanımlanmış olmalıdır. Destek işlevleri olan *girdi_işlevi* ve *çıktı_işlevi* zorunludur. *alış_işlevi*, *gönderim_işlevi* ve *analiz_işlevi* ise isteğe bağlıdır. Genelde bu işlevler C gibi düşük seviyeli bir dille yazılırlar.

girdi_işlevi türün harici metin gösterimini, bu türü kullanmak için tanımlanmış işleç ve işlevler tarafından kullanılan dahili gösterime çevirir. *çıktı_işlevi* ise bunun tersini yapar. Girdi işlevi ya `cstring` türünde tek bir argüman alacak şekilde ya da `cstring`, `oid` ve `integer` türünde üç argüman alacak şekilde bildirilebilir. İlk argüman, bir C dizgesi olarak girdi metnidir. İkinci argüman, tür bir dizi ise dizi elemanlarının nesne kimliği, bir bileşik tür ise türün kendi nesne kimliğidir. Üçüncüsü ise, biliniyorsa, hedef sütunun `typmod`'u, bilinmiyorsa –1'dir. Girdi işlevi yeni veri türünde bir değer ile dönmelidir. Çıktı işlevi ya yeni veri türünde tek bir argüman alacak şekilde ya da ikincisi `oid` türünde iki argüman alacak şekilde bildirilebilir.

İkinci argüman yine, tür bir dizi ise dizi elemanlarının nesne kimliği, bir bileşik tür ise türün kendi nesne kimliğidir. Çıktı işlevi `cstring` türünde bir değer ile dönmelidir.

İsteğe bağlı olan *alış_işlevi* türün harici ikilik gösterimini dahili gösterime çevirir. Eğer bu işlev yoksa, tür ikilik çıktıda rol alamaz. İkilik gösterim oldukça taşınabilir olmanın yanı sıra dahil gösterime dönüşümde ucuz olmalıdır. (Örneğin, standart tamsayı veri türlerinin harici ikilik gösterimleri ağ bayt sıralamasındayken, dahili gösterimleri makinenin doğal bayt sıralamasındadır.) Alış işlevi değerin geçerliliğinden emin olmayı sağlayacak kadar sınama yapmalıdır. Alış işlevi ya `internal` türünde tek bir argüman alacak şekilde ya da `internal` ve `oid` türünde iki argüman alacak şekilde bildirilebilir. İşlev yeni veri türünde bir değer ile dönmelidir. İlk argüman alınan bayt dizgesini tutacak bir `StringInfo` tamponuna bir göstericidir. İkinci argüman, tür bir dizi ise dizi elemanlarının nesne kimliği, bir bileşik tür ise türün kendi nesne kimliğidir. Benzer şekilde, isteğe bağlı olan *gönderim_işlevi* türün dahili gösterimini harici ikilik gösterime çevirir. Gönderim işlevi ya yeni veri türünde tek bir argüman alacak şekilde ya da ikincisi `oid` türünde iki argüman alacak şekilde bildirilebilir. İkinci argüman yine, tür bir dizi ise dizi elemanlarının nesne kimliği, bir bileşik tür ise türün kendi nesne kimliğidir. Gönderim işlevi `bytea` türünde bir değer ile dönmelidir.

Girdi ve çıktı işlevlerinin yeni tür oluşturulmadan önce yeni türde argümanlar ve veri türleri ile bildirimlerinin nasıl yapılacağı noktasında dikkatli olmalısınız. Bunun yanıtı, önce girdi işlevinin sonra da çıktı işlevinin (ve isteniyorsa, ikilik G/Ç işlevlerinin) oluşturulması, son olarak veri türünün tanımlanması olacaktır. PostgreSQL yeni veri türünün ismini ilk defa girdi işlevinin dönüş türü olarak görecek, sistem kataloğunda basitçe yer tutucu girdi olarak bir kabuk türü ayıracak ve girdi işlevinin tanımını kabuk türüne ilintileyecektir. Benzer şekilde diğer işlevleri de (artık mevcut olan) kabuk türüne ilintileyecektir. Son olarak, **`CREATE TYPE`**, kabuk girdisi ile tam tür tanımını yer değiştirir ve yeni tür kullanılabilir olur.

İsteğe bağlı olan *analiz_işlevi*, veri türündeki sütunlar için türe özgü istatistikleri hesaplar. Öntanımlı olarak, **`ANALYZE`** eğer tür için öntanımlı bir b–tree işleç sınıfı varsa, türün eşittir ve küçüktür işleçlerini kullanarak istatistikleri toplamaya çalışacaktır. Bileşik türlerde bu davranış elverişsiz olabilir, bu sorun özel bir analiz işlevi belirterek aşılabilir. Analiz işlevi `internal` türünde tek bir argüman alacak şekilde bildirilmeli ve `boolean` türünde bir sonuçla dönmelidir. Analiz işlevlerinin ayrıntılı uygulama arayüzü `src/include/commands/vacuum.h` dosyasında görülebilir.

Yeni türün dahili gösteriminin ayrıntıları sadece G/Ç işlevleri ve bu türle çalışmak üzere sizin tanımladığınız işlevlerce bilinir; dahili gösteriminin bazı özelliklerinin PostgreSQL'e ayrıca bildirilmesi gerekir. Bunların en önemlisi *dahili_uzunluk*'tur. Temel veri türleri, *dahili_uzunluk* bir pozitif tamsayı olarak verildiğinde sabit uzunlukta olabileceği gibi, *dahili_uzunluk* olarak **`VARIABLE`** belirtildiğinde değişken uzuklukta da olabilir (bu dahili olarak `typlen`'e –1 atanarak yapılır). Değişken uzunluklu tüm veri türleri, türün değerinin toplam uzunluğunu gösteren 4 baytlık bir tamsayı ile başlamalıdır.

İsteğe bağlı olan **`PASSEDBYVALUE`** seçeneği bu veri türünün gösterilerek değil değeri ile aktarılacağını belirtir. Dahili gösterimleri `Datum` türünün genişliğinden (çoğu makinede 4, birkaçında 8 bayttır) daha büyük veri türlerini değeri ile aktaramazsınız.

hizalama parametresi ile belleğin ne uzunlukta adımlanarak veri türünün yerletirileceği belirtilir. İzin verilen adım uzunlukları 1, 2, 4 veya 8 bayttır. Değişken uzunluklu veri türleri için, ilk eleman bir `int4` olduğundan en az 4 baytlık adım uzunluğu belirtilmelidir.

strateji parametresi ile değişken uzunluklu veri türlerinin saklama stratejilerinin seçimi mümkün olur. (Sabit uzunluklu türlerde sadece `plain` mümkündür.) `plain` ile değerin daima olduğu gibi sıkıştırılmadan saklanacağı; `extended` ile, değer çok uzunsa önce sıkıştırılmaya çalışılacağı, yine de uzunsa, ana tablo dışına taşınacağı; `external` ile, değerin ana tablo dışına taşınacağı, fakat sistemin değeri sıkıştırmaya çalışmayacağı; `main` ile, sıkıştırma yapılacağı ama değerin ana tablo dışına taşınmasının engelleneceği belirtilir. `main` saklama stratejisinde, değeri tablo satırı içinde saklamanın bir yolu yoksa, değer yine de

ana tablo dışına taşınabilir, fakat değerin ana tabloda tutulması bakımından böyle bir öğe `extended` ve `external` öğelere göre daha ayrıcalıklıdır.

Kullanıcının sütunlarda NULL değer istememesi durumunda **DEFAULT** seçeneği ile bir öntanımlı değer belirtilebilir. (Böyle bir öntanımlı değer bir sütuna açıkça iliştirilen bir **DEFAULT** ile o sütun için değiştirilebilir.)

Türün bir dizi olduğu, dizi elemanları **ELEMENT** seçeneği kullanılarak belirtilebilir. Örneğin, 4 baytlık tamsayılardan (`int4`) oluşan bir dizi tanımlamak için, seçenek **ELEMENT = int4** şeklinde belirtilir. Dizi türler aşağıda ayrıntılı olarak açıklanmıştır.

Bu türün dizisinin harici gösteriminde kullanmak üzere *ayraç* olarak belli bir karakter belirtilebilir. Öntanımlı ayraç virgüldür. Yalnız, burada belirtilen ayraç dizi türle değil, dizi elemanının türüyle ilgilidir.

Dizi Türler

Bir kullanıcı tanımlı temel veri türünün her oluşturuluşunda, PostgreSQL bu veri türünün dizi türünü kendiliğinden oluşturur ve bu veri türününün ismini temel veri türü isminin başına bir alt çizgi ekleyerek oluşturur. Çözümleyici bu uzlaşımı bilir ve `foo[]` gibi bir türdeki sütun isteklerini `_foo` türündeki isteklere dönüştürür. Dolaylı oluşturulan dizi türü değişken uzunlukludur ve yerleşik girdi ve çıktı işlevleri olan `array_in` ve `array_out` işlevlerini kullanır.

Madem sistem doğru dizi türünü kendiliğinden oluşturuyor, **ELEMENT** diye bir seçenek niçin var diyebilirsiniz. **ELEMENT** kullanmanın yararlı olduğu tek durum, dahili olarak aynı türde şeylerin bir dizisi olmak üzere bir sabit uzunluklu tür tanımlayıp, hem bu türün tamamı üzerinde hem de bu şeylere indisleriyle doğrudan erişerek bazı işlemler yapabilmek istenmesi durumudur. Örneğin, `name` türünün `char` elemanlarına bu yöntemle erişmek mümkündür. Bir iki boyutlu tür olan `point` türünün iki elemanına `point[0]` ve `point[1]` şeklinde erişmek mümkündür. Dahili biçim, eş sabit uzunluklu alanlardan oluştuğundan, bu oluşum sadece sabit uzunluklu türler için geçerlidir. İndislenebilir bir değişken uzunluklu tür, `array_in` ve `array_out` tarafından kullanılan genelleştirilmiş dahili gösterime sahip olmalıdır. Tarihi sebeplerle (bu aslında doğru değil, asıl sebep bunu değiştirmek için geç kalınmış olmasıdır), sabit uzunluklu dizilerin indislemesi sıfırdan başlarken, değişken uzunluklu dizilerde birden başlar.

Parametreler

isim
: Oluşturulacak türün ismi (şema nitelemeli olabilir).

öznitelik_ismi
: Bileşik tür için bir öznitelik (sütun) ismi.

veri_türü
: Bileşik türü oluşturmak üzere bir sütun veri türü olarak mevcut bir türün ismi.

girdi_işlevi
: Türün harici metin gösterimini dahili gösterime çeviren işlevin ismi.

çıktı_işlevi
: Türün dahili gösterimini harici metin gösterimine çeviren işlevin ismi.

alış_işlevi
: Türün harici ikilik gösterimini dahili gösterime çeviren işlevin ismi.

gönderim_işlevi
: Türün dahili gösterimini harici ikilik gösterime çeviren işlevin ismi.

analiz_işlevi

Veri türü için istatistiksel analizler yapan işlevin ismi.

dahili_uzunluk
: Yeni türün dahili gösteriminin bayt cinsinden uzunluğunu belirten sayısal sabit. Öntanımlı değer türün değişken uzunluklu olacağı kabulüne dayanır.

hizalama
: Belleğin ne uzunlukta adımlanarak veri türünün yerletirileceği belirtilir. Belirtilmesi gerekliyse, `char`, `int2`, `int4`, ya da `double` olabilir. `int4` öntanımlıdır.

strateji
: Değişken uzunluklu veri türlerinin saklama stratejisi. Belirtilmesi gerekliyse, `plain`, `external`, `extended` veya `main` olabilir. `plain` öntanımlıdır.

öntanımlı
: Veri türü için öntanımlı değer. Belirtilmezse NULL öntanımlıdır.

öğe_türü
: Belirtilirse türü bir dizi yapar; bu, dizi elemanının veri türü olmalıdır.

ayraç
: Bu türün harici dizi gösteriminde kullanılacak ayraç karakteri.

Ek Bilgi

Kullanıcı tanımlı türlerin isimleri altçizgi (_) karakteri ile başlayamaz ve ençok 62 karakter uzunlukta (veya daha genel olarak, `NAMEDATALEN` – 2; tür ismi dışında bütün isimler için `NAMEDATALEN` – 1) olabilir. Altçizgi ile başlayan tür isimleri dahili olarak oluşturulan dizi tür isimleri için ayrılmıştır.

7.3 öncesi `PostgreSQL` sürümlerinde, işlevlerin `opaque` türde yer tutuculu tür isimlerine ileri başvuruları ile yer değiştirmek üzere bir kabuk türü oluşturmaktan kaçınmak alışılmış bir durumdu. Ayrıca, 7.3 öncesinde, `cstring` argüman ve dönüş türlerinin `opaque` olarak bildirilmeleri zorunluydu. Eski döküm dosyalarını desteklemek için, **`CREATE TYPE`** `opaque` kullanılarak bildirilmiş işlevleri kabul edecek, fakat işlevin bildiriminin doğru tür kullanılacak şekilde değiştirilmesi hususunda bir uyarı çıktılayacaktır.

Örnekler

Bir bileşik türün oluşturulması ve bir işlev tanımında kullanılması:

```
CREATE TYPE compfoo AS (f1 int, f2 text);

CREATE FUNCTION getfoo() RETURNS SETOF compfoo AS $$
    SELECT fooid, fooname FROM foo
$$ LANGUAGE SQL;
```

`box` isminde bir temel veri türünün oluşturulması ve bir tablo tanımında kullanılması:

```
CREATE TYPE box (
    INTERNALLENGTH = 16,
    INPUT = my_box_in_function,
    OUTPUT = my_box_out_function
);

CREATE TABLE myboxes (
    id integer,
```

```
        description box
    );
```

`box` türünün dahili yapısı `float4` türünde 4 elemanlı bir dizi ise:

```
    CREATE TYPE box (
        INTERNALLENGTH = 16,
        INPUT = my_box_in_function,
        OUTPUT = my_box_out_function,
        ELEMENT = float4
    );
```

Bu şekilde, `box` türündeki değerin elemanlarına indisleri ile erişilebileceği gibi, tür yukarıdaki gibi de davranır.

Büyük bir nesne türü oluşturulması ve bir tablo tanımında kullanılması:

```
    CREATE TYPE bigobj (
        INPUT = lo_filein, OUTPUT = lo_fileout,
        INTERNALLENGTH = VARIABLE
    );
    CREATE TABLE big_objs (
        id integer,
        obj bigobj
    );
```

Girdi ve çıktı işlevleri dahil daha fazla örneği,
`http://www.postgresql.org/docs/8.0/static/xtypes.html` adresinde bulabilirsiniz.

Uyumluluk

Bu **`CREATE TYPE`** bir `PostgreSQL` oluşumudur. SQL:1999 ve sonraki standartlarda da bir **`CREATE TYPE`** vardır ama ayrıntıda daha farklıdır.

İlgili Belgeler

`CREATE FUNCTION` [`create_function(7)`], **`DROP TYPE`** [`drop_type(7)`], **`ALTER TYPE`** [`alter_type(7)`].

Çeviren

Nilgün Belma Bugüner <nilgun (at) belgeler·gen·tr>, Nisan 2005

YASAL UYARI

SORUNLAR SİZE AİTTİR. HERHANGİ BİR HATA VEYA EKSİKLİKTEN DOLAYI DOĞABİLECEK OLAN BÜTÜN SERVİS, TAMİR VEYA DÜZELTME MASRAFLARI SİZE AİTTİR.

İLGİLİ KANUNUN İCBAR ETTİĞİ DURUMLAR VEYA YAZILI ANLAŞMA HARİCİNDE HERHANGİ BİR ŞEKİLDE TELİF HAKKI SAHİBİ VEYA YUKARIDA İZİN VERİLDİĞİ ŞEKİLDE BELGEYİ DEĞİŞTİREN VEYA YENİDEN DAĞITAN HERHANGİ BİR KİŞİ, BELGENİN İÇERDİĞİ BİLGİNİN KULLANIMI VEYA KULLANILAMAMASI (VEYA VERİ KAYBI OLUŞMASI, VERİNİN YANLIŞ HALE GELMESİ, SİZİN VEYA ÜÇÜNCÜ ŞAHISLARIN ZARARA UĞRAMASI VEYA BİLGİNİN BAŞKA BİLGİLERLE UYUMSUZ OLMASI) YÜZÜNDEN OLUŞAN GENEL, ÖZEL, DOĞRUDAN YA DA DOLAYLI HERHANGİ BİR ZARARDAN, BÖYLE BİR TAZMİNAT TALEBİ TELİF HAKKI SAHİBİ VEYA İLGİLİ KİŞİYE BİLDİRİLMİŞ OLSA DAHİ, SORUMLU DEĞİLDİR.

PostgreSQL CREATE TYPE(7)

Bu dosya (man7-create_type.pdf), belgenin XML biçiminin TEXLive ve belgeler-xsl paketlerindeki araçlar kullanılarak PDF biçimine dönüştürülmesiyle elde edilmiştir.

31 Ocak 2007