New in v2.0:计算列通过列定义中包含的标量表达式存储从其他列生成的数据。

为什么要使用计算列？

与分区，JSONB列或二级索引一起使用时，计算列特别有用。

* 分区要求使用作为主键前缀的列定义分区。 在地理分区的情况下，某些应用程序将希望折叠此列中可能值的数量，以使某类的查询更具性能。 例如，如果users表具有country和state列，则可以使用简化域创建存储的计算列位置，以用于分区。 有关更多信息，请参阅下面的分区示例。
* JSONB列用于存储半结构化JSONB数据。 当表的主要信息存储在JSONB中时，索引JSONB文档的特定字段很有用。 特别是，计算列允许以下用例：具有PRIMARY KEY列和payload列的双列表，其主键计算为来自payload列的某些字段。 这减轻了手动将主键与JSON blob分离的需要。 有关更多信息，请参阅下面的JSONB示例。
* 二级索引可以在计算列上创建，这在经常对表进行排序时尤其有用。 请参阅下面的二级索引示例。

注意事项

计算列：

* 创建表后无法添加。 请关注GitHub问题以获取有关此限制的更新。
* 不能用于生成其他计算列。
* 不能是外键引用。
* 表现像任何其他列一样，但不能直接写入。
* 与DEFAULT互斥。

创建

只能在创建表时添加计算列。 使用以下语法：

**column\_name** <**type**> **AS** (<**expr**>) **STORED**

| **Parameter** | **Description** |
| --- | --- |
| column\_name | 计算列的名称/标识符。 |
| <type> | 计算列的数据类型。 |
| <expr> | 用于计算列值的纯scalar表达式。不能使用标记为impure的任何函数，例如now()或nextval()。 |
| STORED | （必需）计算列与其他列一起存储。 |

示例

使用计算列创建表

在这个例子中，让我们创建一个带有计算列的简单表：

**CREATE** **TABLE** **names** (

id INT **PRIMARY** **KEY**,

first\_name **STRING**,

last\_name **STRING**,

full\_name **STRING** **AS** (**CONCAT**(first\_name, ' ', last\_name)) STORED

);

然后，插入几行数据：

**INSERT** **INTO** **names** (id, first\_name, last\_name) **VALUES**

(1, 'Lola', 'McDog'),

(2, 'Carl', 'Kimball'),

(3, 'Ernie', 'Narayan');

**SELECT** \* **FROM** **names**;

+----+------------+-------------+----------------+

| id | first\_name | last\_name | full\_name |

+----+------------+-------------+----------------+

| 1 | Lola | McDog | Lola McDog |

| 2 | Carl | Kimball | Carl Kimball |

| 3 | Ernie | Narayan | Ernie Narayan |

+----+------------+-------------+----------------+

full\_name列是从first\_name和last\_name列计算的，无需定义视图。

使用地理分区和计算列创建表

地理分区是一项企业功能。 申请企业版license，查阅[Enterprise Licensing](http://doc.cockroachchina.baidu.com/#deploy/enterprise-licensing/)

在这个例子中，让我们创建一个包含地理分区和计算列的表：

**CREATE** **TABLE** user\_locations (

locality **STRING** **AS** (**CASE**

**WHEN** country **IN** ('ca', 'mx', 'us') **THEN** 'north\_america'

**WHEN** country **IN** ('au', 'nz') **THEN** 'australia'

**END**) STORED,

id SERIAL,

name **STRING**,

country **STRING**,

**PRIMARY** **KEY** (locality, id))

**PARTITION** **BY** LIST (locality)

(**PARTITION** north\_america **VALUES** **IN** ('north\_america'),

**PARTITION** australia **VALUES** **IN** ('australia'));

然后，插入几行数据：

**INSERT** **INTO** user\_locations (name, country) **VALUES**

('Leonard McCoy', 'us'),

('Uhura', 'nz'),

('Spock', 'ca'),

('James Kirk', 'us'),

('Scotty', 'mx'),

('Hikaru Sulu', 'us'),

('Pavel Chekov', 'au');

**SELECT** \* **FROM** user\_locations;

+---------------+--------------------+---------------+---------+

| locality | id | name | country |

+---------------+--------------------+---------------+---------+

| australia | 333153890100609025 | Uhura | nz |

| australia | 333153890100772865 | Pavel Chekov | au |

| north\_america | 333153890100576257 | Leonard McCoy | us |

| north\_america | 333153890100641793 | Spock | ca |

| north\_america | 333153890100674561 | James Kirk | us |

| north\_america | 333153890100707329 | Scotty | mx |

| north\_america | 333153890100740097 | Hikaru Sulu | us |

+---------------+--------------------+---------------+---------+

locality列是从country列计算的。 然后可以将位置值用于地理分区。

使用JSONB列和计算列创建表

在这个例子中，让我们创建一个包含JSONB列和计算列的表：

**CREATE** **TABLE** student\_profiles (

id **STRING** **PRIMARY** **KEY** **AS** (**profile**->>'id') STORED,

**profile** JSONB

);

然后，插入几行数据：

INSERT INTO student\_profiles (profile) VALUES

('{"id": "d78236", "name": "Arthur Read", "age": "16", "school": "PVPHS", "credits": 120, "sports": "none"}'),

('{"name": "Buster Bunny", "age": "15", "id": "f98112", "school": "THS", "credits": 67, "clubs": "MUN"}'),

('{"name": "Ernie Narayan", "school" : "Brooklyn Tech", "id": "t63512", "sports": "Track and Field", "clubs": "Chess"}');

**SELECT** \* **FROM** student\_profiles;

+--------+---------------------------------------------------------------------------------------------------------------------+

| id | profile |

+--------+---------------------------------------------------------------------------------------------------------------------+

| d78236 | {*"age"*: *"16"*, *"credits"*: 120, *"id"*: *"d78236"*, *"name"*: *"Arthur Read"*, *"school"*: *"PVPHS"*, *"sports"*: *"none"*} |

| f98112 | {*"age"*: *"15"*, *"clubs"*: *"MUN"*, *"credits"*: 67, *"id"*: *"f98112"*, *"name"*: *"Buster Bunny"*, *"school"*: *"THS"*} |

| t63512 | {*"clubs"*: *"Chess"*, *"id"*: *"t63512"*, *"name"*: *"Ernie Narayan"*, *"school"*: *"Brooklyn Tech"*, *"sports"*: *"Track and Field"*} |

+--------+---------------------------------------------------------------------------------------------------------------------+

主键id被计算为profile列中的字段。

在计算列上创建具有二级索引的表

在这个例子中，让我们创建一个包含计算列和该列索引的表：

**CREATE** **TABLE** gymnastics (

id **UUID** **PRIMARY** **KEY** **DEFAULT** gen\_random\_uuid(),

athlete **STRING**,

vault DECIMAL,

bars DECIMAL,

beam DECIMAL,

**floor** DECIMAL,

combined\_score DECIMAL **AS** (vault + bars + beam + **floor**) STORED,

**INDEX** total (combined\_score **DESC**)

);

然后，插入几行数据：

**INSERT** **INTO** gymnastics (athlete, vault, bars, beam, **floor**) **VALUES**

('Simone Biles', 15.933, 14.800, 15.300, 15.800),

('Gabby Douglas', 0, 15.766, 0, 0),

('Laurie Hernandez', 15.100, 0, 15.233, 14.833),

('Madison Kocian', 0, 15.933, 0, 0),

('Aly Raisman', 15.833, 0, 15.000, 15.366);

**SELECT** \* **FROM** gymnastics;

+--------------------------------------+------------------+--------+--------+--------+--------+----------------+

| id | athlete | vault | bars | beam | floor | combined\_score |

+--------------------------------------+------------------+--------+--------+--------+--------+----------------+

| 3fe11371-6a6a-49de-bbef-a8dd16560fac | Aly Raisman | 15.833 | 0 | 15.000 | 15.366 | 46.199 |

| 56055a70-b4c7-4522-909b-8f3674b705e5 | Madison Kocian | 0 | 15.933 | 0 | 0 | 15.933 |

| 69f73fd1-da34-48bf-aff8-71296ce4c2c7 | Gabby Douglas | 0 | 15.766 | 0 | 0 | 15.766 |

| 8a7b730b-668d-4845-8d25-48bda25114d6 | Laurie Hernandez | 15.100 | 0 | 15.233 | 14.833 | 45.166 |

| b2c5ca80-21c2-4853-9178-b96ce220ea4d | Simone Biles | 15.933 | 14.800 | 15.300 | 15.800 | 61.833 |

+--------------------------------------+------------------+--------+--------+--------+--------+----------------+

现在，让我们使用二级索引运行查询：

**SELECT** athlete, combined\_score **FROM** gymnastics **ORDER** **BY** combined\_score **DESC**;

+------------------+----------------+

| athlete | combined\_score |

+------------------+----------------+

| Simone Biles | 61.833 |

| Aly Raisman | 46.199 |

| Laurie Hernandez | 45.166 |

| Madison Kocian | 15.933 |

| Gabby Douglas | 15.766 |

+------------------+----------------+

最高综合得分为61.833的运动员是Simone Biles。

See Also

* [Scalar Expressions](http://doc.cockroachchina.baidu.com/#develop/sql-syntax/scalar-expressions/)
* [Information Schema](http://doc.cockroachchina.baidu.com/#develop/information-schema/)
* [CREATE TABLE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/CREATE-TABLE/)
* [JSONB](http://doc.cockroachchina.baidu.com/#develop/data-types/JSONB/)
* [Define Table Partitions (Enterprise)](http://doc.cockroachchina.baidu.com/#deploy/define-table-partitions-(enterprise)/)