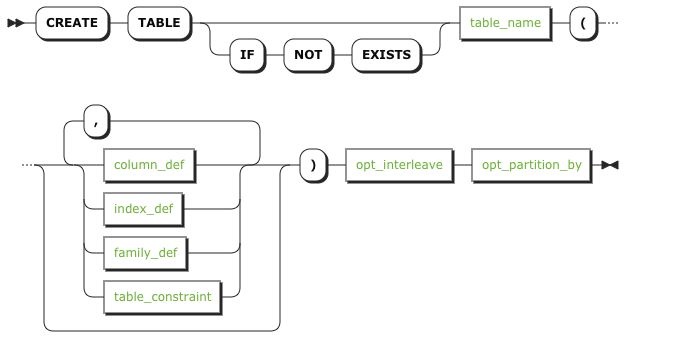
CREATE TABLE用于在数据库创建新表。

所需权限

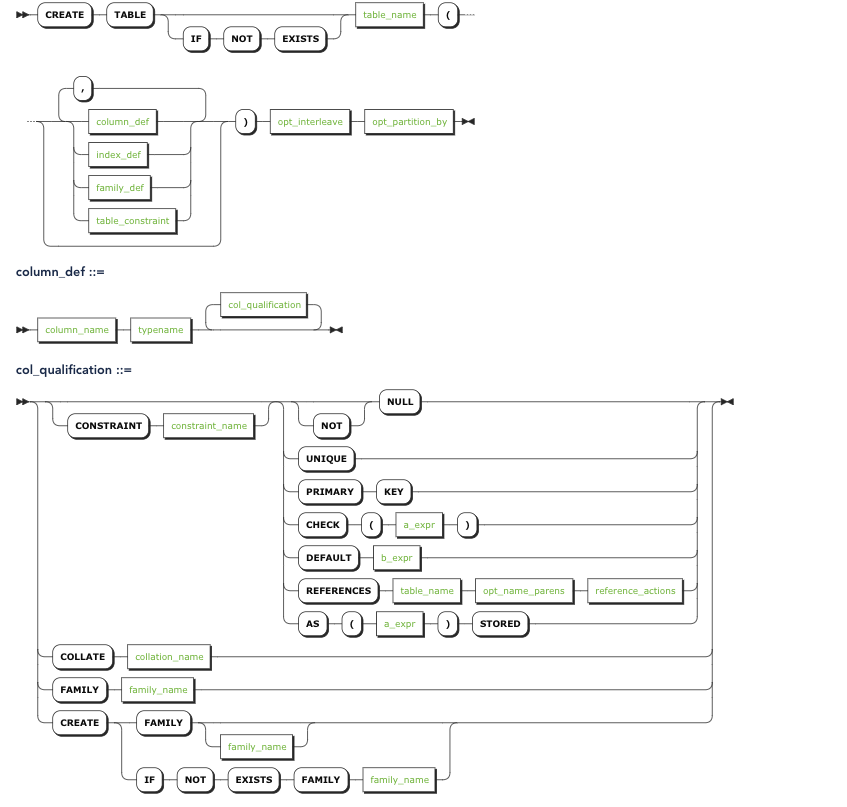
用户必须具有父数据库的CREATE权限。

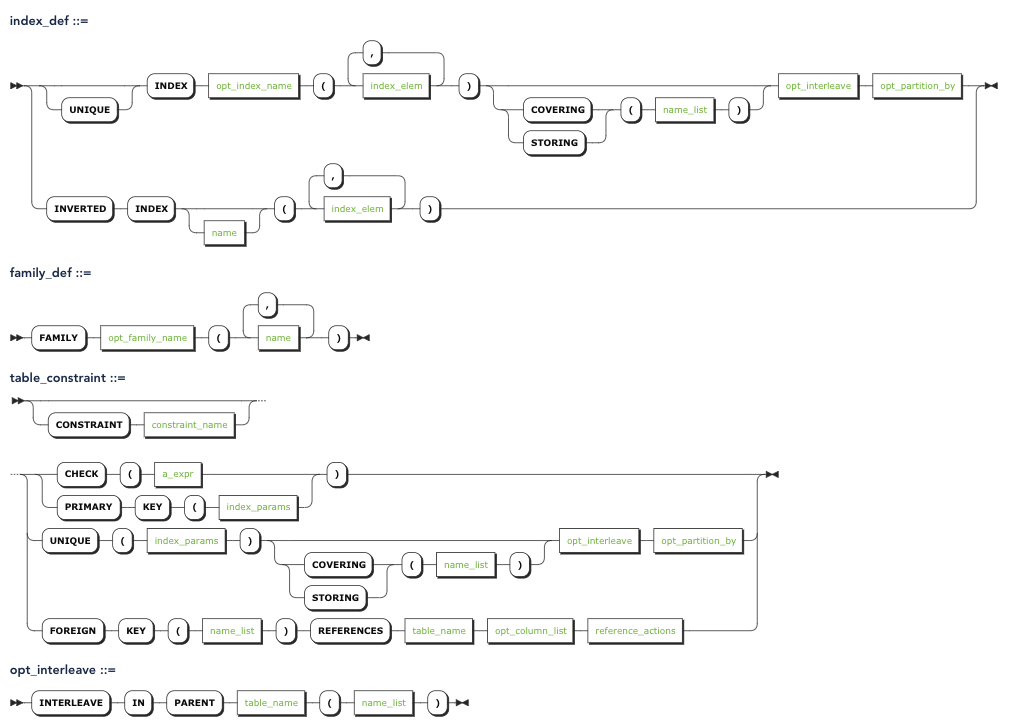
概要

基础



扩展





基于SELECT语句的结果创建表，请使用CREATE TABLE AS。

参数

| **Parameter** | **Description** |
| --- | --- |
| IF NOT EXISTS | 仅当数据库中不存在同名表时才创建新表; 如果确实存在，则不返回错误。  请注意，IF NOT EXISTS仅检查表名; 它不检查现有表是否具有新表的相同列，索引，约束等。 |
| table\_name | 要创建的表的名称，在其数据库中必须是唯一的，并遵循[identifier rules](http://doc.cockroachchina.baidu.com/#develop/sql-syntax/keywords-and-identifiers/#identifiers). 如果没有设置默认父数据库，则必须将名称格式设置为database.name。  UPSERT和INSERT ON CONFLICT语句使用名为excluded的临时表来处理执行期间的唯一性冲突。 因此，不建议对任何表使用名称excluded。 |
| column\_def | 定义列的逗号分隔列表。 每列需要名称/标识符和数据类型; 列级约束或其他列限定（例如计算列）的指定是可选项。 列名在表中必须是唯一的，但可以与索引或约束具有相同的名称。  在列级别定义的任何主键，唯一和检查约束将作为表创建的一部分移动到表级别。 使用SHOW CREATE TABLE语句在表级别查看它们。 |
| index\_def | 可选项，定义索引的逗号分隔列表。 对于每个索引，必须指定要索引的列; 可选择指定名称。 索引名称在表中必须是唯一的，并遵循标识符规则。 请参阅下面的创建具有辅助索引和倒排索引的表示例。  CREATE INDEX语句可用于创建索引，与建表分开。 |
| family\_def | 可选项，定义列族的逗号分隔列表。 列族名称在表中必须是唯一的，但可以与列，约束或索引具有相同的名称。  列族是一组列，它们作为单个键值对存储在基础键值存储中。 CockroachDB自动将列分组到列族中，以确保有效的存储和性能。 但是，有时你可能希望手动将列分配给族，更多详细信息，请参见列族文档。 |
| table\_constraint | 可选项，表级约束的逗号分隔列表。 约束名称在表中必须是唯一的，但可以与列，列族或索引具有相同的名称。 |
| opt\_interleave | 你可以通过交错表来优化查询性能，这会更改CockroachDB存储数据的方式。 |
| opt\_partition\_by | New in v2.0: 一个仅限企业版的选项，允许你在行级别定义表分区。 你可以按列表或按range定义表分区。 有关更多信息，请参阅定义表分区文章。 |

表级复制

默认情况下，表在默认复制区域中创建，但可以放入特定的复制区域。 有关更多信息，请参阅[为表创建复制区](http://doc.cockroachchina.baidu.com/#deploy/configure-replication-zones/#create-a-replication-zone-for-a-table)。

行级复制（New in v2.0）

CockroachDB允许企业用户定义表分区，从而提供行级控制数据的存储方式和位置。 有关更多信息，请参见[为表分区创建复制区](http://doc.cockroachchina.baidu.com/#deploy/configure-replication-zones/#create-a-replication-zone-for-a-table-partition-new-in-v2-0)。

分区所需的主键与传统的主键不同。 要定义用于分区的主键，将主键中的唯一标志符作为前缀，按照你想要嵌套子分区的顺序，添加到你想要进行分区和子分区的表的所有列。 有关详细信息，请参阅使用[主键进行分区](http://doc.cockroachchina.baidu.com/#deploy/define-table-partitions-(enterprise)/#partition-using-primary-key)。

示例

创建表（没有定义主键）

在CockroachDB中，每个表都需要一个主键。 如果未明确定义，则会自动添加INT类型的名为rowid的列作为主键，并使用unique\_rowid()函数确保新行始终默认为唯一的rowid值。 自动为主键创建索引。

严格地说，主键的唯一索引并没有创建; 它由数据存储层的key来区分，因此不需要额外的空间。 但是，在使用SHOW INDEX等命令时，它会显示为正常的唯一索引。

> **CREATE** **TABLE** logon (

user\_id INT,

logon\_date DATE

);

> **SHOW** **COLUMNS** **FROM** logon;

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

| Field | Type | Null | Default | Indices |

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

| user\_id | INT | true | NULL | {} |

| logon\_date | DATE | true | NULL | {} |

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

(2 rows)

> **SHOW** **INDEX** **FROM** logon;

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

| Table | Name | Unique | Seq | Column | Direction | Storing | Implicit |

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

| logon | primary | true | 1 | rowid | ASC | false | false |

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

(1 row)

创建表（定义主键）

在此示例中，我们创建一个包含三列的表。 一列是主键，另一列是唯一约束，第三列没有约束。 具有唯一约束的主键和列将自动创建索引。

> **CREATE** **TABLE** logoff (

user\_id INT **PRIMARY** **KEY**,

user\_email **STRING** **UNIQUE**,

logoff\_date DATE

);

> **SHOW** **COLUMNS** **FROM** logoff;

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

| Field | Type | Null | Default | Indices |

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

| user\_id | INT | false | NULL | {primary,logoff\_user\_email\_key} |

| user\_email | STRING | true | NULL | {logoff\_user\_email\_key} |

| logoff\_date | DATE | true | NULL | {} |

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

(3 rows)

> **SHOW** **INDEX** **FROM** logoff;

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

| Table | Name | Unique | Seq | Column | Direction | Storing | Implicit |

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

| logoff | primary | true | 1 | user\_id | ASC | false | false |

| logoff | logoff\_user\_email\_key | true | 1 | user\_email | ASC | false | false |

| logoff | logoff\_user\_email\_key | true | 2 | user\_id | ASC | false | true |

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

(3 rows)

创建具有二级索引和倒排索引的表（New in v2.0）

在此示例中，我们在表创建期间创建两个二级索引。 二级索引允许使用除主键之外的键有效地访问数据。 此示例还演示了许多列级和表级约束。

倒排索引是v2.0中新增的功能，允许有效访问JSONB列中的无schema数据。

此示例还演示了许多列级和表级约束：

> **CREATE** **TABLE** product\_information (

product\_id INT **PRIMARY** **KEY** **NOT** NULL,

product\_name **STRING**(50) **UNIQUE** **NOT** NULL,

product\_description **STRING**(2000),

category\_id **STRING**(1) **NOT** NULL **CHECK** (category\_id **IN** ('A','B','C')),

weight\_class INT,

warranty\_period INT **CONSTRAINT** valid\_warranty **CHECK** (warranty\_period **BETWEEN** 0 **AND** 24),

supplier\_id INT,

product\_status **STRING**(20),

list\_price DECIMAL(8,2),

min\_price DECIMAL(8,2),

catalog\_url **STRING**(50) **UNIQUE**,

date\_added DATE **DEFAULT** **CURRENT\_DATE**(),

misc JSONB,

**CONSTRAINT** price\_check **CHECK** (list\_price >= min\_price),

**INDEX** date\_added\_idx (date\_added),

**INDEX** supp\_id\_prod\_status\_idx (supplier\_id, product\_status),

INVERTED **INDEX** details (misc)

);

> **SHOW** **INDEX** **FROM** product\_information;

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

| Table | Name | Unique | Seq | Column | Direction | Storing | Implicit |

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

| product\_information | primary | true | 1 | product\_id | ASC | false | false |

| product\_information | product\_information\_product\_name\_key | true | 1 | product\_name | ASC | false | false |

| product\_information | product\_information\_product\_name\_key | true | 2 | product\_id | ASC | false | true |

| product\_information | product\_information\_catalog\_url\_key | true | 1 | catalog\_url | ASC | false | false |

| product\_information | product\_information\_catalog\_url\_key | true | 2 | product\_id | ASC | false | true |

| product\_information | date\_added\_idx | false | 1 | date\_added | ASC | false | false |

| product\_information | date\_added\_idx | false | 2 | product\_id | ASC | false | true |

| product\_information | supp\_id\_prod\_status\_idx | false | 1 | supplier\_id | ASC | false | false |

| product\_information | supp\_id\_prod\_status\_idx | false | 2 | product\_status | ASC | false | false |

| product\_information | supp\_id\_prod\_status\_idx | false | 3 | product\_id | ASC | false | true |

| product\_information | details | false | 1 | misc | ASC | false | false |

| product\_information | details | false | 2 | product\_id | ASC | false | true |

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

(12 rows)

关于索引还可阅读以下资料：

* 使用[CREATE INDEX](https://www.cockroachlabs.com/docs/stable/create-index.html)为现有表创建索引。
* [更多关于索引的内容](http://doc.cockroachchina.baidu.com/#develop/performance-optimization/indexes/).

使用自动生成的唯一行ID创建表

要自动生成唯一行ID，请使用带有gen\_random\_uuid()函数的UUID列作为默认值：

CREATE TABLE t1 (id UUID PRIMARY KEY DEFAULT gen\_random\_uuid(), name STRING);

INSERT INTO t1 (name) VALUES ('a'), ('b'), ('c');

SELECT \* FROM t1;

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

| id | name |

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

| 60853a85-681d-4620-9677-946bbfdc8fbc | c |

| 77c9bc2e-76a5-4ebc-80c3-7ad3159466a1 | b |

| bd3a56e1-c75e-476c-b221-0da9d74d66eb | a |

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

(3 rows)

或者，可以使用带有uuid\_v4()函数的BYTES列作为默认值：

CREATE TABLE t2 (id BYTES PRIMARY KEY DEFAULT uuid\_v4(), name STRING);

INSERT INTO t2 (name) VALUES ('a'), ('b'), ('c');

SELECT \* FROM t2;

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

| id | name |

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

| "**\x**9b**\x**10**\xdc\x**11**\x**9a**\x**9cGB**\xbd\x**8d**\t\x**8c**\xf**6@vP" | a |

| "**\xd**9s**\xd**7**\x**13**\n**\_L\***\xb**0**\x**87c**\xb**6d**\xe**1**\xd**8@" | c |

| "**\uac**74**\x**1dd@B**\x**97**\xac\x**04N&**\x**9eBg**\x**86" | b |

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

(3 rows)

在任何一种情况下，生成的ID都是128位，足够大所以几乎不会生成重复值。 此外，一旦表增长超出单个kv range（默认情况下超过64MB），新ID将分散在所有表的range内，因此可能跨越不同的节点。 这意味着多个节点将共担负载。

如果有必要将生成的ID存储在相同的kv range内，可以使用SERIAL数据类型，它是INT的别名，默认值是unique\_rowid()：

**CREATE** **TABLE** t3 (id SERIAL **PRIMARY** **KEY**, name **STRING**);

**INSERT** **INTO** t3 (name) **VALUES** ('a'), ('b'), ('c');

**SELECT** \* **FROM** t3;

+*--------------------+------+*

| id | name |

+*--------------------+------+*

| 293807573840855041 | a |

| 293807573840887809 | b |

| 293807573840920577 | c |

+*--------------------+------+*

(3 rows)

在insert时，unique\_rowid()函数从执行insert的节点的时间戳和ID生成默认值。 除非在每个节点每秒生成非常大量的ID（100,000+）的情况下，否则这种时间排序的值可认为是全局唯一的。

创建具有外键约束的表

外键约束保证列仅使用它引用的列中已存在的值，该列必须来自另一个表。 此约束强制实现两个表之间的引用完整性（referential integrity）。

管理外键有许多规则，但最重要的两个是：

* 使用INDEX，PRIMARY KEY或UNIQUE创建表时，必须索引外键列。
* 引用列必须仅包含唯一值。 这意味着REFERENCES子句必须使用与主键或唯一约束完全相同的列。

New in v2.0: 你可以包含一个外键操作，来当引用的列发生更新或删除时执行的操作。 默认操作是ON UPDATE NO ACTION和ON DELETE NO ACTION。

在这个例子中，我们使用ON DELETE CASCADE（当删除由外键约束引用的行时，也删除所有相关行）。

> **CREATE** **TABLE** customers (

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

name **STRING**

);

> **CREATE** **TABLE** orders (

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

customer\_id INT **REFERENCES** customers(id) **ON** **DELETE** **CASCADE**

);

> **SHOW** **CREATE** **TABLE** orders;

+*--------+---------------------------------------------------------------------------------------------------------------------+*

| Table | CreateTable |

+*--------+---------------------------------------------------------------------------------------------------------------------+*

| orders | **CREATE** **TABLE** orders (␤ |

| | id INT **NOT** NULL,␤ |

| | customer\_id INT NULL,␤ |

| | **CONSTRAINT** "primary" **PRIMARY** **KEY** (id **ASC**),␤ |

| | **CONSTRAINT** fk\_customer\_id\_ref\_customers **FOREIGN** **KEY** (customer\_id) **REFERENCES** customers (id) **ON** **DELETE** **CASCADE**,␤ |

| | **INDEX** orders\_auto\_index\_fk\_customer\_id\_ref\_customers (customer\_id **ASC**),␤ |

| | FAMILY "primary" (id, customer\_id)␤ |

| | ) |

+*--------+---------------------------------------------------------------------------------------------------------------------+*

> **INSERT** **INTO** customers **VALUES** (1, 'Lauren');

> **INSERT** **INTO** orders **VALUES** (1,1);

> **DELETE** **FROM** customers **WHERE** id = 1;

> **SELECT** \* **FROM** orders;

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

| id | customer\_id |

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

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

创建一个KV存储镜像的表

CockroachDB是一个基于事务性和强一致性的KV存储的分布式SQL数据库。 虽然无法直接访问KV存储，但你可以使用两列“简单”表直接镜像访问，其中一组作为主键：

**CREATE** **TABLE** kv (k INT **PRIMARY** **KEY**, v BYTES);

当这样的“简单”表没有索引或外键时，INSERT / UPSERT / UPDATE / DELETE语句转换为开销最小的kv操作（single digit percent slowdowns）。 例如，以下用于在表中添加或替换行的UPSERT将转换为单个键值Put操作：

UPSERT INTO kv VALUES (1, b'hello')

此SQL表方法还为你提供定义明确的查询语言，已知的事务模型，以及在需要时向表中添加更多列的灵活性。

从SELECT语句创建一个表

你可以使用CREATE TABLE AS语句从SELECT语句的结果创建一个新表，例如：

> **SELECT** \* **FROM** customers **WHERE** state = 'NY';

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

| id | name | state |

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

| 6 | Dorotea | NY |

| 15 | Thales | NY |

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

> **CREATE** **TABLE** customers\_ny **AS** **SELECT** \* **FROM** customers **WHERE** state = 'NY';

> **SELECT** \* **FROM** customers\_ny;

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

| id | name | state |

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

| 6 | Dorotea | NY |

| 15 | Thales | NY |

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

使用计算列创建表（ New in v2.0

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。

使用分区创建表（New in v2.0）

分区所需的主键与传统的主键不同。 要定义用于分区的主键，将主键中的唯一标志符作为前缀，按照你想要嵌套子分区的顺序，添加到你想要进行分区和子分区的表的所有列。 有关详细信息，请参阅使用[主键进行分区](http://doc.cockroachchina.baidu.com/#deploy/define-table-partitions-(enterprise)/#partition-using-primary-key)。

按列表创建包含分区的表

在此示例中，我们创建一个表并按列表定义分区：

> **CREATE** **TABLE** students\_by\_list (

id SERIAL,

name **STRING**,

email **STRING**,

country **STRING**,

expected\_graduation\_date DATE,

**PRIMARY** **KEY** (country, id))

**PARTITION** **BY** LIST (country)

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

**PARTITION** australia **VALUES** **IN** ('AU','NZ'),

**PARTITION** **DEFAULT** **VALUES** **IN** (**default**));

使用range创建包含分区的表

在此示例中，我们创建一个表并按range定义分区。

> **CREATE** **TABLE** students\_by\_range (

id SERIAL,

name **STRING**,

email **STRING**,

country **STRING**,

expected\_graduation\_date DATE,

**PRIMARY** **KEY** (expected\_graduation\_date, id))

**PARTITION** **BY** RANGE (expected\_graduation\_date)

(**PARTITION** graduated **VALUES** **FROM** (MINVALUE) **TO** ('2017-08-15'),

**PARTITION** **current** **VALUES** **FROM** ('2017-08-15') **TO** (MAXVALUE));

显示表的定义

要显示表的定义，请使用SHOW CREATE TABLE语句。 CreateTable列的内容是一个包含嵌入换行符的字符串，在回显时会生成格式化输出。

> **SHOW** **CREATE** **TABLE** logoff;

+*--------+----------------------------------------------------------+*

| Table | CreateTable |

+*--------+----------------------------------------------------------+*

| logoff | **CREATE** **TABLE** logoff (␤ |

| | user\_id INT **NOT** NULL,␤ |

| | user\_email **STRING**(50) NULL,␤ |

| | logoff\_date DATE NULL,␤ |

| | **CONSTRAINT** "primary" **PRIMARY** **KEY** (user\_id),␤ |

| | **UNIQUE** **INDEX** logoff\_user\_email\_key (user\_email),␤ |

| | FAMILY "primary" (user\_id, user\_email, logoff\_date)␤ |

| | ) |

+*--------+----------------------------------------------------------+*

(1 **row**)

See Also

* [INSERT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/INSERT/)
* [ALTER TABLE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/ALTER-TABLE/)
* [DELETE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/DELETE/)
* [DROP TABLE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/DROP-TABLE/)
* [RENAME TABLE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/RENAME-TABLE/)
* [SHOW TABLES](http://doc.cockroachchina.baidu.com/#develop/sql-statements/SHOW-TABLES/)
* [SHOW COLUMNS](http://doc.cockroachchina.baidu.com/#develop/sql-statements/SHOW-COLUMNS/)
* [Column Families](http://doc.cockroachchina.baidu.com/#develop/performance-optimization/column-families/)
* [Table-Level Replication Zones](http://doc.cockroachchina.baidu.com/#deploy/configure-replication-zones/#create-a-replication-zone-for-a-table)
* [Define Table Partitions](http://doc.cockroachchina.baidu.com/#deploy/define-table-partitions-(enterprise)/)