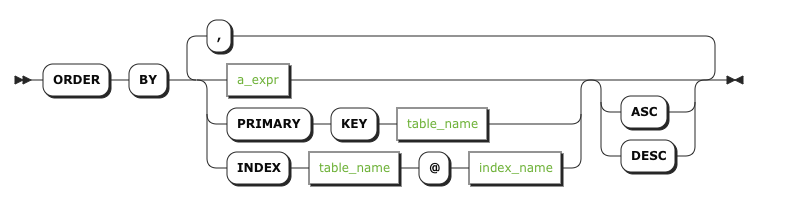
ORDER BY 子句控制返回或处理行的顺序。它可以用在任何查询语句中，包括INSERT或UPSERT的操作数，以及DELETE和UPDATE语句。

Synopsis



参数

ORDER BY子句采用逗号分隔的排序列表。 每个排序规则由选择的列决定，后跟可选的关键字ASC或DESC。

每个**column selection** 可以采用以下形式之一

* 一个简单的列选择，确定如下：
* 在SELECT子句中先前使用AS配置的列标签的名称。 使用SELECT子句计算的值作为排序键。
* 一个正整数，指定数据源中的一列，要么是SELECT子句的FROM子句，要么是DELETE或UPDATE写入的表。 使用来自数据源的相应输入值作为排序键。
* 任意标量表达式。 使用将该表达式计算的结果作为排序键。
* 符号PRIMARY KEY <table\_name>。 这使用给定表的主键列作为排序键。 该表必须是数据源的一部分。
* 符号INDEX <table\_name> @ <index\_name>。 这使用由给定索引索引的列作为排序键。 该表必须是数据源的一部分。

列选择后的可选关键字ASC表示按原样使用排序键，因此没有意义。

可选关键字DESC反转所选列的方向。

Order Preservation

通常，即使指定了ORDER BY，也无法保证查询的中间结果的顺序。换句话说，ORDER BY子句仅在顶层语句中有效。 例如，查询计划程序在FROM子句中的子查询中出现时会忽略它，如下所示：

> **SELECT** \* **FROM** a, b **ORDER** **BY** a.x; *-- valid, effective*

> **SELECT** \* **FROM** (**SELECT** \* **FROM** a **ORDER** **BY** a.x), b; *-- ignored, ineffective*

但是，将查询与子查询组合在一起时，某些组合会使子查询中的ORDER BY子句变得有意义：

1. 保留WITH ORDINALITY子句的操作数的顺序（在SELECT子句的FROM操作数内），以控制行的编号。 2.保留独立LIMIT或OFFSET子句（在SELECT子句的FROM操作数内）的操作数的顺序，以确定在结果中保留哪些行。
2. 保留INSERT语句或也使用LIMIT的UPSERT语句的数据源的顺序，以确定处理哪些行。
3. 保留标量表达式中使用的子查询的顺序。

例如，使用WITH ORDINALITY：

> **SELECT** \* **FROM** (**SELECT** \* **FROM** a **ORDER** **BY** a.x) **WITH** ORDINALITY;

*-- 确保行按列a.x的顺序编号。*

例如，在FROM中使用独立的LIMIT子句：

> **SELECT** \* **FROM** a, ((**SELECT** \* **FROM** b **ORDER** **BY** b.x) **LIMIT** 1);

*-- 确保只按b.x列的顺序排列第一行b*

*-- 用于交叉连接。*

例如，在INSERT中使用LIMIT：

> **INSERT** **INTO** a (**SELECT** \* **FROM** b **ORDER** **BY** b.x) **LIMIT** 1;

*-- 确保只按b.x列的顺序排列第一行b*

*-- 被插入a。*

例如，在scalar上下文中使用子查询：

> **SELECT** ARRAY(**SELECT** a.x **FROM** a **ORDER** **BY** a.x);

*-- 确保使用排序顺序的a.x值构造数组。*

> **SELECT** ARRAY[1, 2, 3] = ARRAY(**SELECT** a.x **FROM** a **ORDER** **BY** a.x);

*-- 确保右侧的值按列a.x的顺序进行比较。*

没有ORDER BY的行排序

如果没有ORDER BY，则以非确定性顺序处理或返回行。 “非确定性”意味着实际的顺序可能取决于逻辑计划，磁盘上的数据顺序，CockroachDB集群的拓扑结构，并且通常随时间变化。

使用简单列选择进行排序

考虑下表：

> **CREATE** **TABLE** a(a INT);

> **INSERT** **INTO** a **VALUES** (1), (3), (2);

以下语句是等效的：

> **SELECT** a **AS** b **FROM** a **ORDER** **BY** b; *-- first form: refers to an AS alias.*

> **SELECT** a **FROM** a **ORDER** **BY** 1; *-- second form: refers to a column position.*

> **SELECT** a **FROM** a **ORDER** **BY** a; *-- third form: refers to a column in the data source.*

+---------+

| a |

+---------+

| 1 |

| 2 |

| 3 |

+---------+

(3 rows)

请注意，规则的顺序很重要。 如果存在歧义，则AS别名优先于数据源列，例如：

> **CREATE** **TABLE** ab(a INT, b INT);

> **SELECT** a **AS** b, b **AS** c **FROM** ab **ORDER** **BY** b; *-- orders by column a, renamed to b*

> **SELECT** a, b **FROM** ab **ORDER** **BY** b; *-- orders by column b*

也可以使用为每行计算的任意scalar表达式进行排序，例如：

> **SELECT** a, b **FROM** ab **ORDER** **BY** a + b; *-- orders by the result of computing a+b.*

使用多列排序

当给出多个排序规则时，后面的规则用于排序与前面规则相同的行，例如：

> **CREATE** **TABLE** ab(a INT, b INT);

> **SELECT** a, b **FROM** ab **ORDER** **BY** b, a;

这将按列b对结果进行排序，然后如果在b列中有多个行具有相同的值，则它将按列a对这些行进行排序。

倒序排序

可以在排序规则之后添加关键字“DESC”（“降序”）以反转其顺序。 这可以针对每个规则单独指定，例如：

> **CREATE** **TABLE** ab(a INT, b INT);

> **SELECT** a, b **FROM** ab **ORDER** **BY** b **DESC**, a; *-- sorts on b descending, then a ascending.*

按主键顺序排序

ORDER BY PRIMARY KEY保证结果以主键顺序呈现。

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

> **SELECT** k, v **FROM** kv **ORDER** **BY** **PRIMARY** **KEY** kv; *-- guarantees ordering by column k.*

如果主键已经使用关键字DESC，那么如果ORDER BY子句也使用DESC，则其含义将被翻转（取消），例如：

> **CREATE** **TABLE** ab(a INT, b INT, **PRIMARY** **KEY** (b **DESC**, a **ASC**));

> **SELECT** \* **FROM** ab **ORDER** **BY** b **DESC**; *-- b按降序递减，然后按升序递减。*

*-- 主索引可用于优化。*

> **SELECT** \* **FROM** ab **ORDER** **BY** **PRIMARY** **KEY** ab **DESC**; *-- 按升序排序，然后降序排列。*

*-- 索引顺序被反转。*

按索引顺序排序

ORDER BY INDEX保证结果以给定索引的顺序呈现。

一个特别的优点是，对于使用该索引的查询，这可以保证顺序，同时还保证不会有额外的排序计算来实现它，例如：

> **CREATE** **TABLE** kv(k INT **PRIMARY** **KEY**, v INT, **INDEX** v\_idx(v));

> **SELECT** k, v **FROM** kv **ORDER** **BY** **INDEX** kv@v\_idx; *-- 保证按列v排序。*

如果索引已经使用关键字DESC，那么如果ORDER BY子句也使用DESC，则其含义将被翻转（取消），例如：

> **CREATE** **TABLE** ab(a INT, b INT, **INDEX** b\_idx (b **DESC**, a **ASC**));

> **SELECT** \* **FROM** ab **ORDER** **BY** b **DESC**; *-- b按降序递减，然后按升序递减。*

*-- 索引b\_idx可用于优化。*

> **SELECT** \* **FROM** ab **ORDER** **BY** **INDEX** ab@b\_idx **DESC**; *-- 按升序排序，然后降序排列。*

*-- 索引顺序被反转。*

在聚合时处理排序

CockroachDB当前以非确定性顺序处理聚合（例如，SELECT ... GROUP BY）。

对于大多数聚合函数，如MIN，MAX，COUNT，无论如何顺序都无关紧要，因为函数是可交换的并且无论如何都会产生相同的结果。 但是，对于少数不可交换的聚合函数（例如，array\_agg()，json\_agg()和concat\_agg()），这意味着聚合的结果将不是确定性的。

这是将来可能会解决的[已知限制](https://github.com/cockroachdb/cockroach/issues/23620)。

See Also

* [Selection Queries](http://doc.cockroachchina.baidu.com/#develop/sql-syntax/selection-queries/)
* [Scalar Expressions](http://doc.cockroachchina.baidu.com/#develop/sql-syntax/scalar-expressions/)
* [INSERT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/INSERT/)
* [UPSERT](http://doc.cockroachchina.baidu.com/#develop/sql-statements/UPSERT/)
* [DELETE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/DELETE/)
* [UPDATE](http://doc.cockroachchina.baidu.com/#develop/sql-statements/DELETE/)