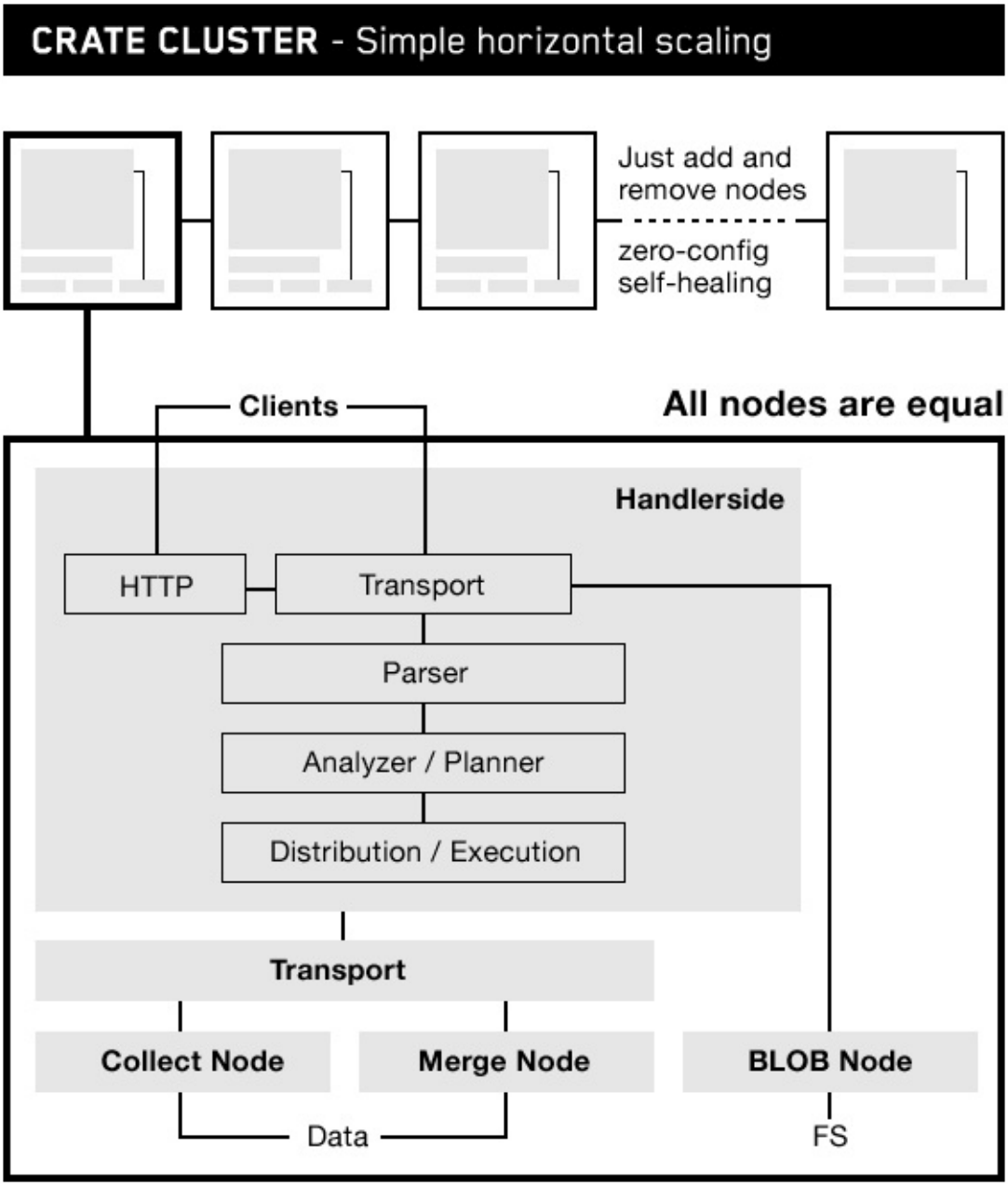
Table of Contents

crate中文文档

为了做好Crate的实践，现打算将其官方文档翻译成中文，由于第一次做技术翻译，翻译的不准确的地方请大家指正。由于Crate版本更新比较快，现在翻译的是0.56.3版本，后续随着版本更新，会相应地更新。

CrateDB基于NoSQL的架构，但是有标准SQL的特性。它是一个无共享的分布式的数据库，支持文档和动态模式的关系。它安装和使用起来非常简单，有着自动分片，自动分区，自动复制的功能。实现了实时搜索和聚合，而且有通过部署CrateDB来水平扩展。它提供写后一致性，平衡内存磁盘的使用量而且是微服务(例如Docker)的理想选择。CrateDB是一个开源的数据库而且在Apache 2.0许可之下。



索引: 默认情况下，CrateDB对所有字段进行索引，将数据存储在列中，这些字段针对过滤和聚合做了一些优化。表上不需要加锁以便添加新列甚至嵌套对象。

无主节点:一个CrateDB 集群是一个无主的，有一组对等节点。可以部署在任意地方:在笔记本上，部署在私有云和共有云上。你可以最大化性价比在廉价的通用的服务器上部署，而且任然可以获取较快较高的性能。

为微服务制作:使用官方的Docker容器允许你快速而且简单地部署CrateDB节点。CrateDB可以在GCE(谷歌的云)上一键部署，而且AWS上有官方的亚马逊的镜像。

更多的架构和技术细节，阅读此技术概览。

可扩展的

CrateDB 已经部署在生产环境中，具有以下规模:

- 每天有数十亿的插入和更新(同时提供面向用户的实时查询)
- 100+的节点(AWS和内部部署)
- 100多TB的数据
- 每秒150多万条的插入(每个节点每秒4万条)
- 10万用户的并发下每秒支持的请求更多(实践的例子：每秒330,000的插入(每个1KB)下,同时面向用户服务,全文查询时延小于400毫秒,运行在8个商用服务器(每个$2000),有64G内存，使用SSD硬盘。
- 云服务提供商的多个可用区域

crate提供现代NoSql世界的优点，无主节点，可扩展，以管理，快速-这与关系型世界相反。然而，它的SQL查询引擎意味着你可以像在任何关系型数据库一样执行实时查询，但是没有关系型数据库的大开销而且更快速。

SQL-over-hadoop解决了相同的声明，但是实际上hadoop上有一个事务层。Crate不是那样的，它是一个无主节点的数据存储提供一个分布式的，使用标准SQL查询语言的查询引擎。

	Crate	**RDBMS**
Data	Structured and semi/unstructured	Structured
ACID	Eventually consistent, best for simple transactions	ACID compliant & fully transactional
Schemas	Dynamic	Fixed/defined
Scaling	Horizontal scaling on commodity hardware: massive growth, no data purging	Scale-up - with growing usage, requires purging and maintenance
Query speeds	Scalable read/write - super fast	Require ops and ETL
Availability	Fully transparent - highly available with automatic replication Commodity hardware	Moderate/high, relies on expensive hardware and ops
Change	Agile	Difficult

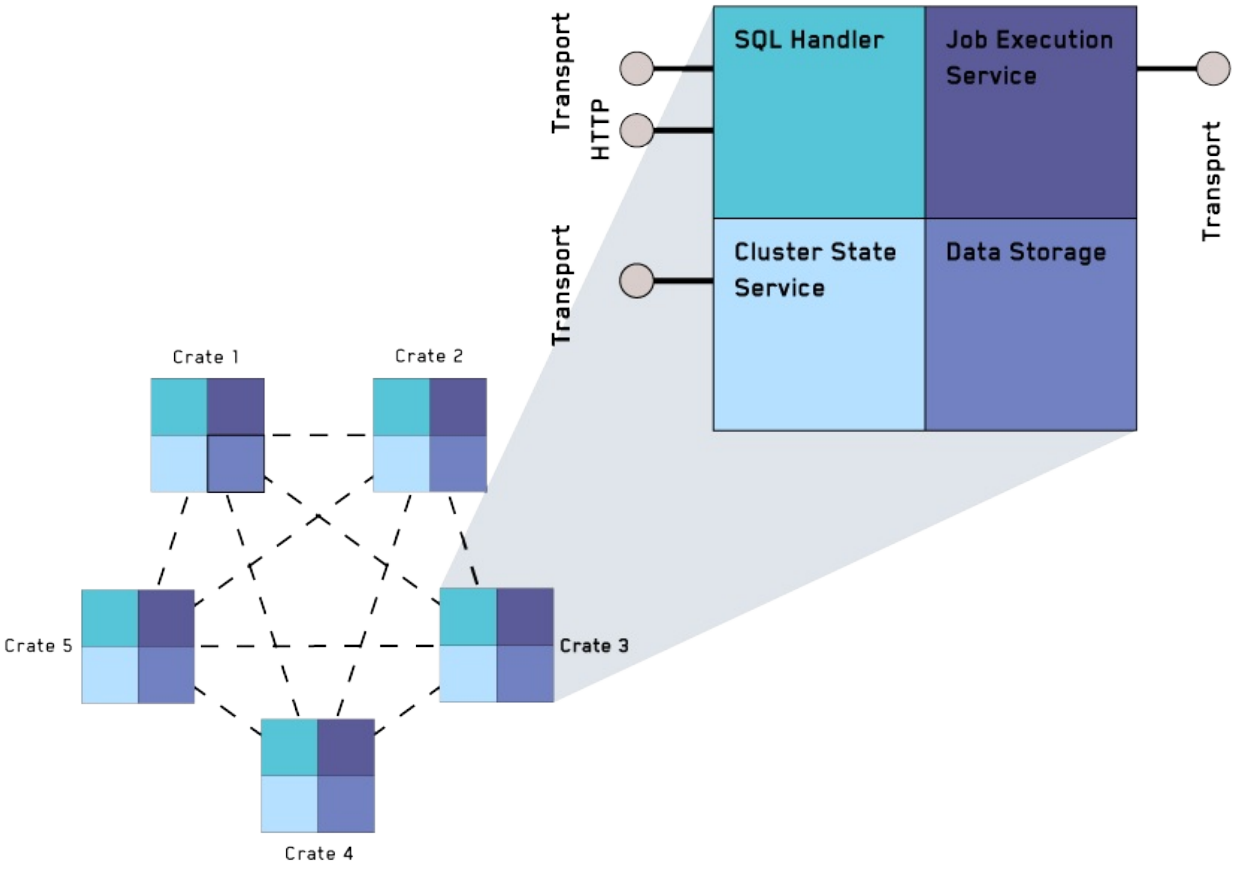
集群和无共享

本文的目的是在高层次上描述分布式数据库Crate如何使用无共享架构形成高可用性弹性的数据库集群。将展示Crate五共享架构的核心概念。它与基于控制器的架构(例如主/从架构)的主要区别是，Crate集群的每个节点都可以执行每个操作，因此所有节点在功能上都是对等的，并配置相同。

Components of a Crate Node(Crate节点的组件)

要理解Crate集群是如何工作的，首先查看集群的单个节是由哪些组件组成的。

多个内部的Crate实例形成单个集群数据库。每个节点的组件是相等的。



[图1]

图 1 显示集群内的每个Crate节点和其他节点一样都有相同组件 (a) 节点之间的接口 (b) 有和其他节点有相同的组件 (c) and/or和外面世界 这些主要的四个组件是:SQL Handler(SQL执行引擎), Job Execution Service(任务执行服务), Cluster State Service(集群状态服务),和Data Storage(数据存储)。

SQL Handler(SQL处理器)

节点的SQL处理器负责三个方面的职责: (a) 处理客户接入的请求 (b) 解析和分析SQL语句 (c) 基于分析的语句创建执行计划(抽象语法树) SQL处理器仅仅有四个组件通过接口和外界通讯。Crate支持两种协议来处理客户端的请求:(a)HTTP (b)二进制传输协议。一个特殊的请求包括一个SQL语句和他人的交互参数。

Job Execution Service(作业执行服务)

作业执行服务负责执行一个计划(“job”任务)。作业的执行步骤和操作结果已经在一个计划中定义。一个作业通常包括多个操作，同过传输协议分发到涉及到的节点，或许是本地的节点并且/或者 一个或多个远程节点。作业主要包括各自的操作的IDS。允许Crate"跟踪"分布式的查询。

Cluster State Service(集群转态服务)

集群转态服务的三个主要的功能: (a) 集群转态管理 (b) 元数据主节点的选举 (c) 节点的发现 因此它是集群主要的组件(如多节点设置:Clusters)。它通过二进制的传输协议。

Data Storage(数据存储)

数据存数组件处理基于执行计划的存储数据和检索数据的操作。在Crate中，存储在表中的数据是分片的，这意味着表被通过多个节点分片来存储。每一个分片是一个独立的Lucene的索引物理地存储在文件系统中。读和写都是在一个分片的级别来操作。

Multi Node Setup: Clusters(多节点安装：集群)

一个Crate集群是不同的主机上组两个或多个Crate实例，这些实例是一个单个的数据库单元。节点之间交互通过Crate特殊的软件传输协议来序列化java普通对象(POJOS)而且在一个独立的端口操作。被叫做“transport port”应该开启，并且可以被集群访问。

Cluster State Management(集群状态管理)

集群状态是版本化的而且所有在一个集群中的节点保持集群最新状态的一个副本。然而，仅仅一个在集群中的单节点-“主元数据节点”-允许在运行期间改变状态。这个节点由集群中所有节点选举产生。

Settings, Metadata, Routing配置,元数据，路由

集群的状态包括所有必须的元数据的信息来维护集群并协调操作。

- Global cluster settings(全局集群配置)
- Discovered nodes and their status(发现节点和它们的状态)
- Schemas of tables(表模式)
- The status and location of primary and replica shards(主分片和副本分片的位置和状态)

当主节点更新集群状态时，它将发布新的状态到集群中的所有节点而且等待所有节点回应在下个更新之前。

Primary Election(主节点的选择)

无共享架构经常称作“无主节点”架构。对于Crate，在数据存储方面是正确的选择。但是协调元数据，设置和DDL需要动态选择主元数据节点。在Crate集群中任何节点都有资格被选为主元数据节点，但如果需要也可以将其限制在节点的一个子集中，每个集群都只能有一个单一的元数据主节点。

为了确保在部分网络(集群的一部分节点不可用)中集群需要仲裁才能选择元数据主节点。因此，仲裁必须大于集群中预期节点的一半。

q = n / 2 + 1

因此如果6个节点的集群被分成两半，则每个分区将不能选择主节点，因为将不会满足4个节点定义的定额。这防止集群形成两个单独的主，不可避免地变得不同步并且导致集群”脑裂“。可以在运行期设置所需的仲裁。

Discovery(节点发现)

查找，添加和删除节点过程都是在节点发现模块中完成的。

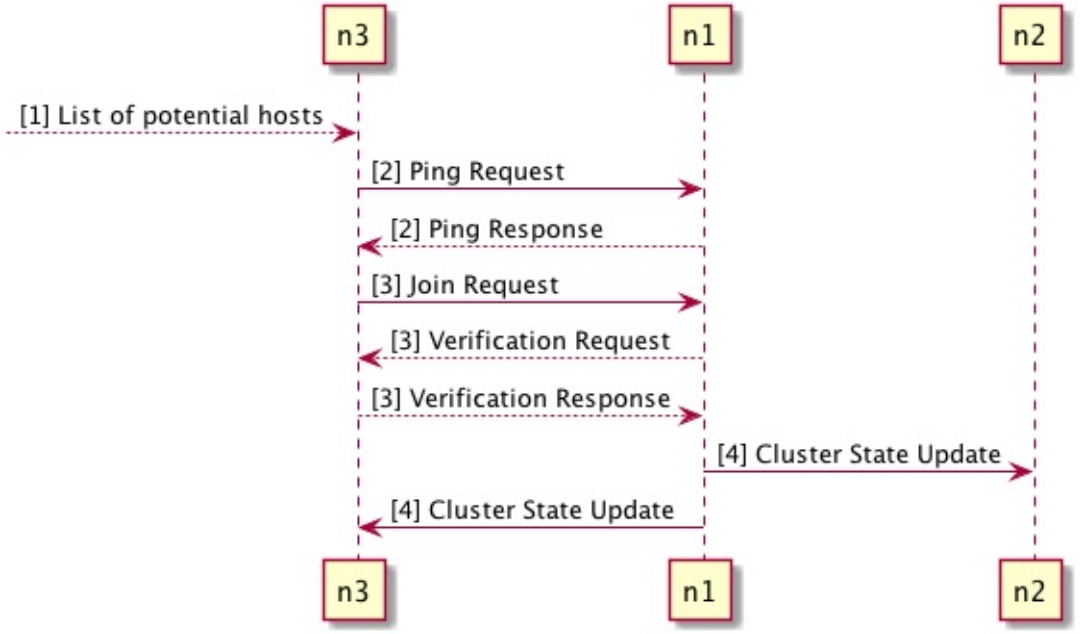


图2

节点发现过程的阶段。n1和n2已经形成一个集群，而且n1被选为主节点，n3加入集群。集群状态的更新并行发生。 节点发现有多个步骤: 在单播的情况下，Crate在启动时需要其他Crate实例潜在的主机/IP地址的列表。该列表可以由静态配置提供或者被动态地生成，例如通过获取DNS SRV记录，查询EC2的API等。

在多播情况下不需要该列表，因为步骤2将发送多播ping请求。

收到请求的节点将回应它应属于的集群信息，当前的集群的主节点和它自己的名字。

现在节点知道元数据主节点，它将发送一个加入集群的请求。主节点验证接入的请求并且向现在包含集群中所有节点完整列表的集群状态中添加一个新的节点。

集群状态通过集群发布。这保证了节点添加是集群中的公共的信息。

Networking(网络)

在一个Crate集群中所有节点都具有到其他节点的直接链路，这被称为全网拓扑。为了简单，每个节点维持网络中每个其他节点的单向链接。5节点的集群的网络拓扑图如下所示:

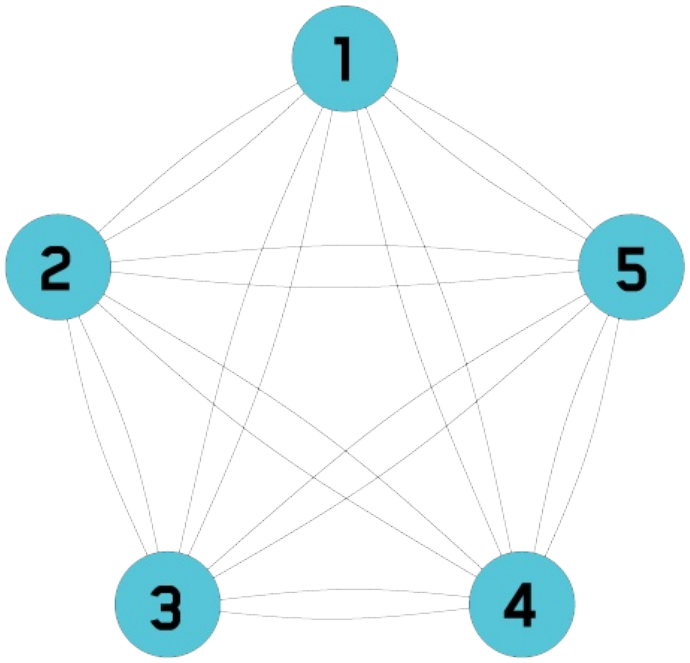


图 3

5节点的Crate集群的网络拓扑。每条线出现一个单向的连接。

全网连接的优点是它提供高度的可靠性，并且节点之间的路径是尽可能最短的。然而这种网络的应用的大小是有限制的，因为连接数(c)随着节点数(n)二次增长。 c = n * (n - 1)

Cluster Behavior(集群行为)

集群中的每个Crate节点平等的事实允许应用程序和用户连接到任何节点，并获得相同的操作的相同响应。眞如在在Crate组件一节中讲的SQL 处理器负责使用http或其他传输协议来处理传入客户端SQL请求。接收客户端请求的处理程序也将响应返回给客户端。它不会将请求重定向或委派给不同的节点。处理程序节点将传入的请求解析为语法树，对其进行分析并在本地建立执行计划。然后以分布式方式执行计划的操作。执行的最后阶段的上游总是处理程序，然后处理程序将响应返回给客户端。

Application Use Case(

应用程序用例)

在使用主/从数据库的应用程序的常规设置中，部署栈如下图所示:

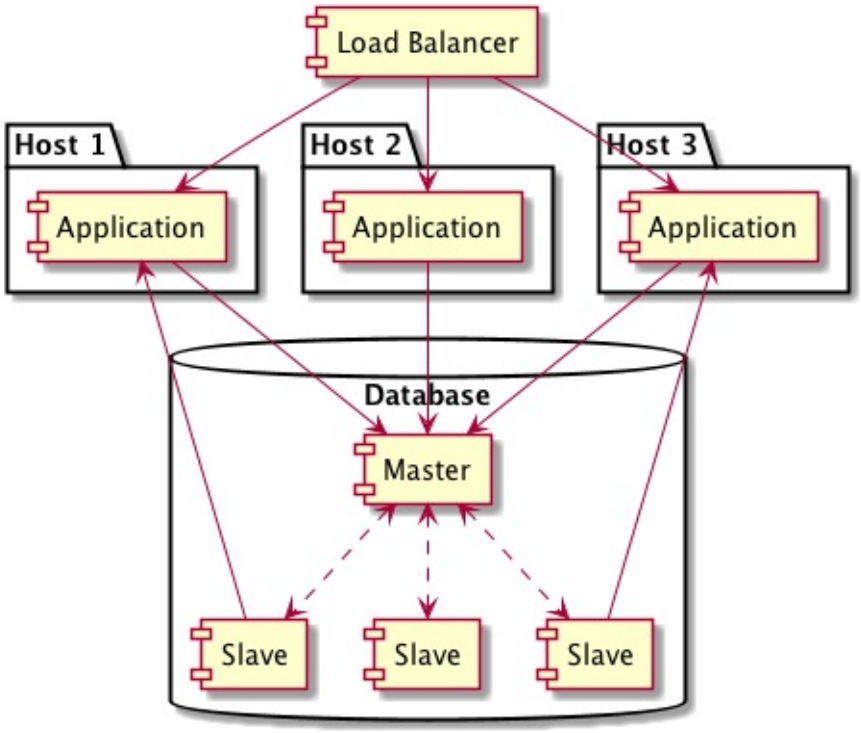


图 4 应用程序部署栈

然而，给出的安装不能扩展，因为所有的应用服务器使用的都是相同的单个数据入口点供数据库写入(应用程序任然可以从从节点读取)而且如果入口点不可用整个栈将不可用。

选在一个无共享架构允许DevOps以无单点故障的“弹性”方式部署它们的应用。无共享的点子从数据库到应用程序，因为在大多数情况下已经无状态。

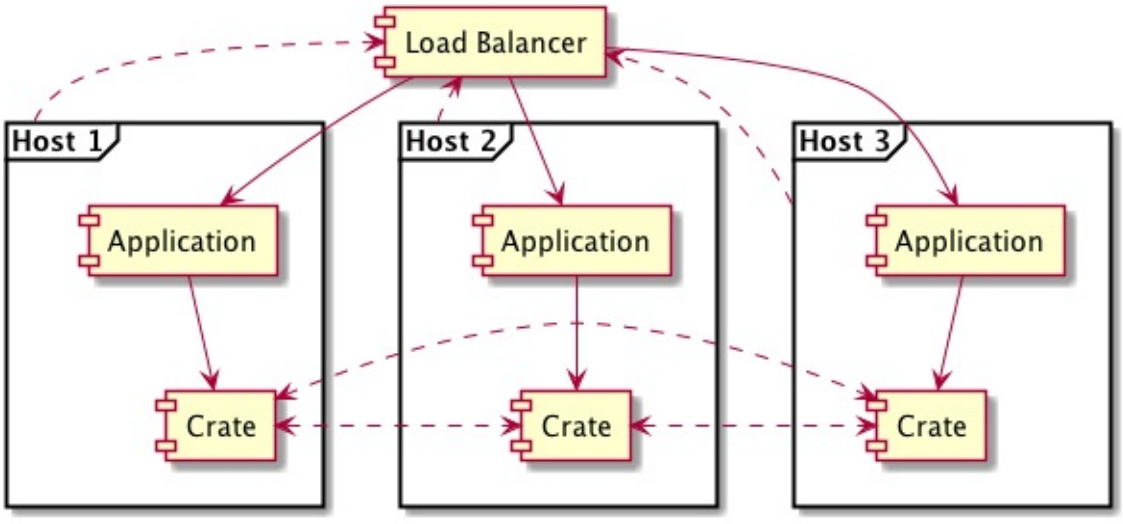


图 5

弹性部署利用无共享架构。

如果你将Crate实例和每个应用服务一起部署，你将能够根据需要动态扩展和缩小数据库后端。这些应用程序只需要与localhost上绑定的Crate实例通信。负载平衡将跟踪主机的运行状态，如果单个主机上的应用程序或者数据库发生故障，则整个主机将从负载平衡中移除。

JOINS(表连接)

JOINs are essential operations in relational databases. They create a link between rows based on common values and allow the meaningful combination of these rows. Crate supports joins and due to its distributed nature allows you to work with large amounts of data.

表连接在关系型数据库中是很有用的操作。他们创建一个行和公共值之前的一个连接而且允许这些行有意义的组合。Crate支持连接而且由于Crate天然支持分布式，所以它可以支持大数据量的操作。

In this document we will present the following topics. First, an overview of the existing types of joins and algorithms provided. Then a description of how Crate implements them, and finally how Crate uses optimizations to work with huge datasets.

在这个文档中我们将阻止以下的主题。 首先，一个已经存在的连接类型和提供的算法。

Types of Joins(连接类型)

A join is a relational operation that merges two data sets based on certain properties. Figure 1 (Inspired by this article) shows which elements appear in which join.

一个连接的关系型操作可以基于确切的属性合并两个数据集。(图1)展示哪些元素将在连接中出现。

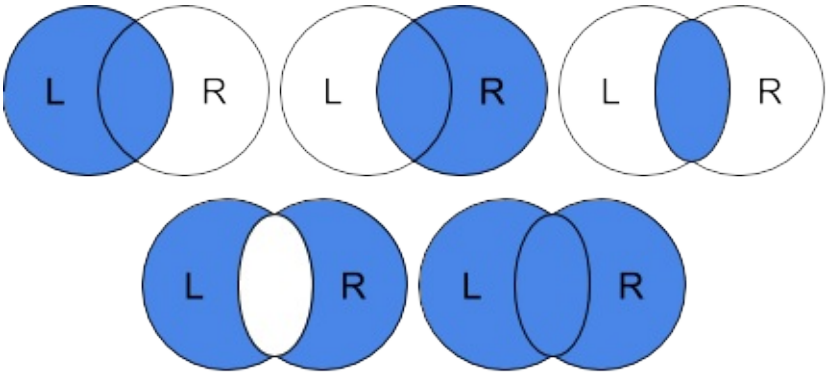


图1

左连接，右连接，内连接，外连接，而且L和R的集合的交集

Cross Join(交叉连接)

返回一个或多个关系的笛卡尔积。L和R关系的笛卡尔积的结果是有L关系中所元组和R关系中元组的所有可能的排列组合。

Inner Join(内连接)

An inner join is a join of two or more relations that returns only tuples that satisfy the join condition.

一个内连接是一个两个或者多个的关系的连接而且只返回满足连接条件的元组。

Equi Join(等值连接)

An equi join is a subset of an inner join and a comparison-based join, that uses equality comparisons in the join condition. The equi join of the relation L and R combines tuple lof relation L with a tuple r of the relation R if the join attributes of both tuples are identical.

一个等值连接是一个内部连接和一个比较连接的子集，在连接关系中使用等值连接。关系L和关系R的等值连接是把L和R通过有相等的属性值的元组来连接。

Outer Join(外连接)

An outer join returns a relation consisting of tuples that satisfy the join condition and dangling tuples from both or one of the relations, respectively to the outer join type.

一个外连接返回满足连接条件的行组成的关系，以及从两个或其中一个关系到外部链接类型的悬空行。

An outer join has following types:(一个外连接有以下类型)

Left outer join returns tuples of the relation L matching tuples of the relation R and dangling tuples of the relation R padded with null values.

左外连接返回关系L中匹配到关系R中的元组。用空值填充R关系中的空值元组。

Right outer join returns tuples of the relation R matching tuples of the relation L and dangling tuples from the relation L padded with null values.

右外连接返回关系R中匹配到L关系中的元组，而且将L中空的元组用空值填充。

Full outer join returns matching tuples of both relations and dangling tuples produced by left and right outer joins.

全外连接返回左外连接和右外连接以及填充的空值元组。

Joins in Crate(Crate中的连接)

Crate supports (a) CROSS JOIN, (b) INNER JOIN, (c) EQUI JOIN, (d) LEFT JOIN, (e) RIGHT JOIN and (f) FULL JOIN. To implement these, the nested loop join algorithm is implemented with a few optimizations.

Crate支持(a)交叉连接,(b)内连接，(c)等值连接，(d)左连接，(e)右连接和(f) 全连接。为了实现这些，使用一系列内部循环连接算法。

Nested Loop Join(内部循环连接)

The nested loop join is the simplest join algorithm. One of the relations is nominated as the inner relation and the other as the outer relation. Each tuple of the outer relation is compared with each tuple of the inner relation and if the join condition is satisfied, the tuples of the relation L and R are concatenated and added into the new relation:

嵌套循环连接是最简单的连接算法。其中一个关系被指定为内部关系，另一个被指定为外部关系。外关系的每个元组和每一个内部关系的每个元组进行比较，并且如果满足条件，则关系L和R的元组被连接并添加到新的关系中:

```
for each tuple l ∈ L do
    for each tuple r ∈ R do
        if l.a θ r.b
            put tuple(l, r) in Q
```

Listing 1. Nested loop join algorithm. 列表 1. 内部循环连接算法 Other Algorithms 其他算法。

Sort-Merge Join and Hash Join are currently not implemented. More information can be found here.

排序-合并连接和散列连接当前还没有实现。更多的信息请查看这里 Pimitive Nested Loop(原始嵌套循环)

For joins on some relations, the nested loop operation can be executed directly on the handler node. Specifically for queries involving a CROSS JOIN or joins on system tables/information_schema each shard sends the data to the handler node. Afterwards, this node runs the nested loop, applies limits, etc. and ultimately returns the results. Similarly, joins can be nested, so instead of collecting data from shards the rows can be the result of a previous join or table function.

对于一些关系连接，循环嵌套操作将被在每个处理节点上执行。特别为涉及到CROSS JOIN和连接到system tables/information_schema的查询，每个分片将发送数据到处理节点。

Distributed Nested Loop(分布式的内循环)

Relations are usually distributed to different nodes which require the nested loop to acquire the data before being able to join. After finding the locations of the required shards (which is done in the planning stage), the smaller data set (based on the row count) is broadcast amongst all the nodes holding the shards they are joined with. After that, each of the receiving nodes can start running a nested loop on the subset it has just received. Finally, the results are pushed to the original (planner) node to merge and return the results to the requesting client (see Figure 2).

关系通常分布到不同的节点，这些节点需要嵌套循环以在能够加入之前获取数据。 找到需要的分片的位置后，最小的数据集(基于行计数)在所有持有它需要连接的分片集群节点间广播。之后，每一个收到的节点可以在它们接收到的子集上运行循环嵌套。最终,结果将会推送到原始(执行计划)节点合并并将结果返回给请求的客户端。

If the rows in the join result from or in another (nested) join or use a table function , the data is broadcast from (or to) different nodes directly.

如果在连接中的行来自其他的内部链接或者使用一个表函数，这些数据将直接来自于广播或者广播出去。

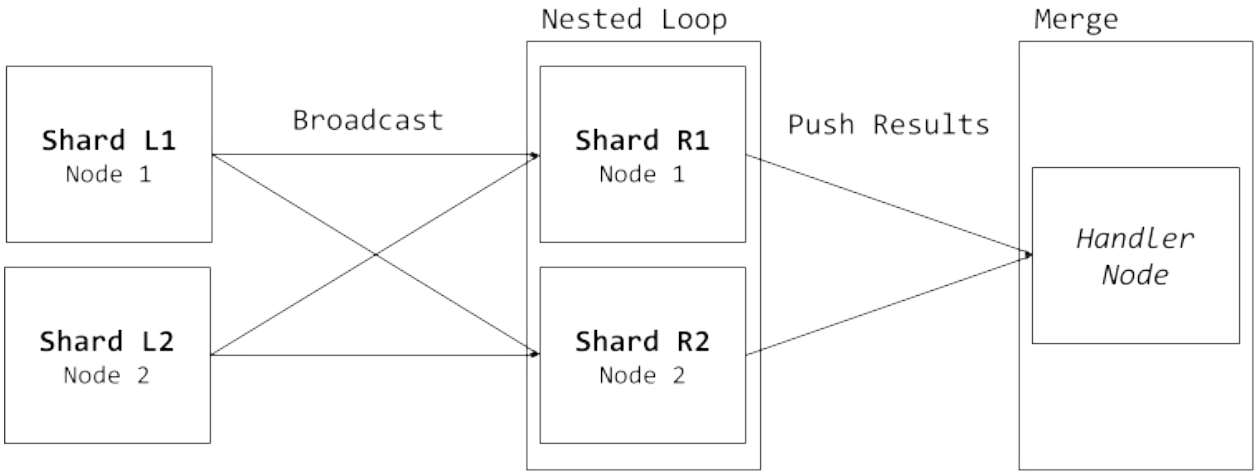


图 2

Nodes that are holding the smaller shards broadcast the data to the processing nodes which then return the results to the requesting node.

> 注意持有的小的数据分片关闭数据到处理节点，这些节点将结果返回请求的节点。

Optimization(优化)

Crate implements joins using a nested loop - which means that the runtime complexity grows exponentially (O(n*m)). Specifically for Cross Joins this results in large amounts of data sent over the network and loaded into memory at the handler node. Crate reduces the volume of data transferred by employing Query Then Fetch: First, filtering and ordering are applied (if possible where the data is located) to obtain the affected document IDs. Next, as soon as the final data set is ready, Crate fetches the selected fields and returns the data to the client.

Crate使用嵌套循环实现连接-这意味着运行时间复杂度将指数级增长。在处理节点连接，大数据量的结果的交集发送到网络并且加载到处理节点的内存中。Crate 通过查询在获取来减少传输的数据量：首先，应用过滤和排序来使用获得文档的IDS。然后,一旦最终数据集已经准备好，Crate将取出选择好的字段的数据返回给客户端。

Pre-Ordering and Limits(在排序和限制数据条数之前)

Queries can be optimized if they contain (a) ORDER BY, (b) LIMIT, or (c) if INNER/EQUI JOIN. In any of these cases, the nested loop can be terminated earlier:

查询可以被优化如果他们包含 (a) ORDER BY, (b) LIMIT 或者 (c) INNER/EQUI JOIN。在这些情况下，嵌套循环可能被打断。

- Ordering allows determining whether there are records left
- Limit states the maximum number of rows that are returned

Consequently, the number of rows is significantly reduced allowing the operation to complete much faster.

- 排序取决于连接的左边是否有记录
- limit表示返回的最大行数

因此，行数越少，操作完成越快。

Push-down Query Optimization(查询优化)

Complex queries such as Listing 2 require the planner to decide when to filter, sort, and merge in order to efficiently execute the plan. In this case, the query would be split internally into subqueries before running the nested loop. As shown in Figure 3, first filtering (and ordering) is applied to relations L and R on their shards, then the result is directly broadcast to the nodes running the nested loop. Not only will this behavior reduce the number of rows to work with, it also distributes the workload among the nodes so that the (expensive) join operation can run faster.

复杂查询如列表2为了有效执行计划，需要计划者来决定什么时候过滤，排序，而且合并。在这个例子中，在循环嵌套执行前，查询将被内部分割成一些子查询。像在图3中所示，首先过滤(并且排序)将被应用在关系L和R的分片上，之后结果直接广播到执行嵌套循环的节点。这样操作不仅仅会减少工作的行，也可以在节点之间分散负载。因此连接操作将会执行得更快。

```
SELECT L.a, R.x
FROM L, R
WHERE L.id = R.id
  AND L.b > 100
  AND R.y < 10
ORDER BY L.a
```

Listing 2. An INNER JOIN on ids (effectively an EQUI JOIN) which can be optimized.

列表 2.一个在ids上的内部连接(有效的一个等值连接)可以被优化

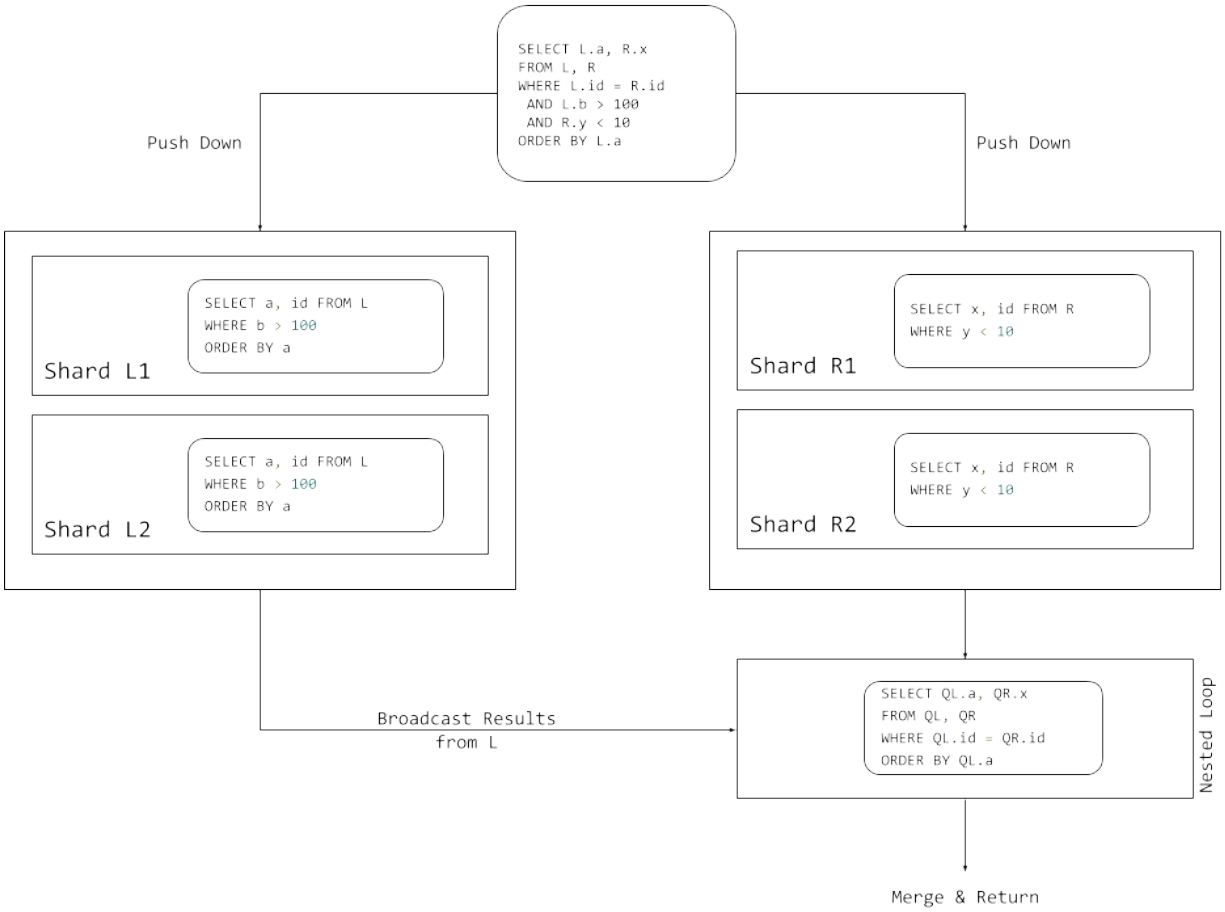


Figure 3

Complex queries are broken down into subqueries that are run on their shards before joining.

存储和一致性

文档提供了一个Crate怎么存储和分布其集群状态的概览，以及一致性和持久性的保证。

> 注意 由于Crate在很大程度上依赖于Elasticsearch和Lucene的存储和集群，Elasticsearch的用户看到这里的概念可能会很眼熟，因为实际上是从ElasticSearch的代码中重用的。

数据存储

Crate中每个表都是分片的，也就说表被切分存储在一个集群的多个节点之间。Crate中的每一个分片时一个Lucene的索引被分成段存储在文件系统上。物理上文件驻留在节点配置的数据目录下。

Lucene仅仅向段文件里追加数据，这意味着写入磁盘的文件将永远不会变。这使得复制和恢复很容易，因为同步分片只是简单地从特定的标记获取数据。

可以为每个表配置任意数量的副本分片。每个可操作的副本都保存一个和主分片完全同步的副本。

对于读操作，在主分片或任何副本分片上执行是没有区别的。Crate当执行路由操作时将随机分配分片。如果有必要可以配置此行为。更多细节，查看我们的多区域最佳实践指南。

写操作和读操作处理完全不同。这些操作在所有活动副本上以如下的方式来同步:

1.给定的操作下，将会在集群状态中查找主分片和活动的副本分片。这一步要成功，主分片和配置的副本分片的仲裁必须可用。

2.操作被路由到相应的主分片以便执行。

3.该操作在主分片上执行。

4.如果操作在主分片上执行成功，则将对所有副本并行执行此操作。

5.在所有副本的操作都执行完了，操作结果将会返回给调用者。

任何副本分片在步骤5中写数据失败后或者写超时，则将会立即被视为不可用。

文档级别的原子性

在Crate中，表的每一行是一个半结构化的文档，可以通过使用对象和数据类型任意嵌套。

对文档的操作都是原子的。这意味着在一个文档上的写操作作为一个整体成功或者没有任何影响。不管文档的嵌套深度的大小如何，都是这种情况。

Crate不提供事务。一旦Crate中的文档被赋予一个版本号，它将在每次变化后增长，所以像乐观并发控制这样的模式(参见使用Crate的乐观并发控制)可以帮助解决这个限制。

持久性

而一个分片有一个预写式日志,也被称为translog。他保证对文档的操作将持久化到磁盘，而不必为每个写操作发出Lucene-Commit操作。当translog被刷新时，所有的数据被写入Lucene的持久索引存储库，并且translog被清除。

在分片不正常关闭的情况下，tranlog中的事务在启动时被重放，以确保所有的操作时永久性的。

当一个新分配的副本从主分片初始化自己时也会直接传送translog。而不需要将段刷新到磁盘仅仅是为了副本分片的恢复。

文档的地址

每个文档都有一个内部的标识(查看 _id)。默认情况下，此标识派生自主键。生成在没有主键的表中，文档在创建时会自动分配一个唯一自动生成的ID。

每个文档使用路由关键字来路由到一个特定的分片。默认情况下这个key是列_id的值。然而这可以在表模式定义中来设置(查看路由)。

虽然对用户透明，但内部有两种方式如何创建已访问的文档:

get:

直接使用标识访问。仅当路由关键字和标识可以从给定的查询规范中计算出来。(了如:在where子句中指定完整主键)

这是访问文档的一个最有效的方法，因为只有一个分片被访问，并且只需要在_id字段上进行简单的索引查找。

search:

通过匹配表的所有候选分片中的文档字段进行查询。

一致性

Crate对于搜索操作最终是一致的。搜索操作在共享的indexReaders上执行，除了其他的功能外，还为碎片提供缓存和反向查找。一个IndexReader总被绑定从它启动的段，这意味着它被刷新才能看到新的更改，这会基于时间的方式也可以手动完成。因此，如果indexReader在变化后刷新，搜索将会看到一个更改。

如果一个查询规范导致在一个get操作，更改立即可见。这是通过首先在translog中查询实现的，该文档将始终具有文档的最新版本。因此，公共更新和获取使用情况是可能的。如果客户端更新一行而且更改后通过其主键查找该行，更改始终可见，因为直接从translog中检索信息。

> 注意
> 每个副本分片和它的主分片同步更新而且存储同样的信息。因此访问主分片和副本分片是一致的。只有刷新IndexReaderaffects的一致性。

集群元数据

集群元数据保存在所谓的“集群状态”中，其中包含以下信息：

- Tables schemas(表的模式)
- 主分片和副本分片的位置。基本上是从分片号到存储节点的映射。
- 每个分片的状态，将指示着分片当前是否已准备好使用，或者有其它状态，例如“初始化”,“恢复”或根本不能分配。
- 有关已发现节点和它们的状态的信息。
- 配置信息。

每个节点都有对集群状态的一份拷贝。然而只有一个节点允许在运行期间改变集群状态。这个节点被称作“主”节点而且是被自动选举出来的。“主”节点没有特殊的配置，任何在集群中节点都可以被选举为主节点。如果当前的主节点由于某种原因宕机也会自动重新选举。

> 注意
> 避免这样一个场景，因为网络分区化而选举出两个主节点，集群需要定义一个仲裁的节点数，用来选举主节点。更多的细节请看主节点的选举。

解释一系列导致集群状态变化的事件-这里有一个改变表模式的“ALTER TABLE”语句：

1.一个在集群中的节点接收到ALTER TABLE的请求。

2.节点发送一个请求到当前集群中的主节点来改变表定义。

3.主节点在本地改变集群状态并且发送一个变化通知给所有影响到的节点。

4.节点收到更改通知，并按此更改修改自己的信息，因此现在所有节点都和主节点的信息是同步的。

5.每个节点可能会执行一些本地的操作这依赖于集群的状态改变的类型。

安装

本章介绍在单节点上安装crate。可在其他支持多播网络的机器上重复此过程，如果想要搭建一个crate集群。这些节点将会相互发现并自动组成一个集群，就像变魔术。

如果网络环境不支持多播，那就需要单播的配置。请参考我们的多节点启动的文章 Multi Node Setup ，来学习如何进行单播的配置。

Java (JVM) version

crate需要工作在java 8的虚拟机上。我们推荐在苹果系统上使用Oracle的java，在linux的系统上使用OpenJDK。必须在crate的所有节点和客户端都使用相同的jvm版本。

我们建议安装 java 8 update 20或者最新的版本。

Install via tar.gz通过tar.gz安装

从https://crate.io/download/下载最新的crate稳定版本，然后解压

```
sh$ tar zxf crate-*.tar.gz
sh$ cd crate-*
```

一旦解压完成，crate可以在前台启动如下:

```
sh$ ./bin/crate
```

为了在后台启动可以加参数 -d 来启动。如果在前台启动的你可以使用Control-C来停止此进程。

Crate脚本

```
sh$ ./bin/crash
```

分布式的crate附带了一个基于web的管理页面。它服务在4200端口上，一旦crate启动,你可以使用你的浏览器访问它。

```
http://host1.example.com:4200/admin/
```

管理平台运行在每一个安装有的crate的节点上。

在docker上安装

Docker是一个轻量级容器可运行分布式应用。方便安装，我们提供为Crate提供Docker镜像。

```
sh$ docker pull crate
sh$ docker run -d -p 4200:4200 -p 4300:4300 crate crate -Des.cluster.name=my_cluster
```

crate镜像托管在Docker的镜像库中。在Github的docker-crate仓库中有一些如何在Docker容器中运行Crate的细节。

为生产环境安装crate

我们为基于RHEL/YUM的系统提供了分布式的版本，如Ubuntu, Debian和 Arch Linux

你可以找到一些怎样安装这些生产环境的crate的说明在下载页面。

如果你的发行版缺少请随时与我们联系。我们很高兴地创建和维护额外的软件包，如果我们看到足够的兴趣。

> 参见
>
> Crate配置
>
> Hello Crate - 学习怎么和Crate交互。
>
> 多节点安装 - 在多个节点上安装和运行Crate。

自从Crate有了合理的默认值，再也不需要为使用基本功能而进行任何配置。

Crate以配置文件配置为主，它位于config/crate.yml。标准的配置文件有默认的有效配置以及注释，可以随着包一起分发。

可以像这样启动时指定配置文件：

```
sh$ ./bin/crate -Des.path.home=/path/to/config/directory
```

任何配置项都可以通过配置文件或系统属性进行配置。如果使用系统属性文件配置，参数前缀'es.(这有个点)'会被忽略。

例如，使用系统属性配置集群的名称将使用如下：

```
sh$ ./bin/crate -Des.cluster.name=cluster
```

这和在配置文件中设置集群名称是一样的。

cluster.name = cluster

这些设置配置文件将按照以下的顺序生效,后面的将会覆盖前面的配置

internal defaults(内部默认值) system properties(系统属性) options from config file(配置文件) command-line properties(命令行)

Table Settings(表设置)

更多的关于如何创建表的语法请参考CREATE TABLE

number_of_replicas(复制份数)

Default: 1
Runtime: yes
指定正常情况下每个表的每个分片的复制份数

refresh_interval

Default: 1000
Runtime: yes
指定每个分片的刷新间隔，以毫秒为单位

Blocks

blocks.read_only

Default: false
Runtime: yes
若果设置为true，表将会只读，而且不允许写，更新和删除操作。这和设置blocks.read和blocks.metadata这两个属性为true效果是一样的。

blocks.read

Default: false
Runtime: yes
若果设置为true，将不允许对此表进行读操作(包括做导出和快照)。

blocks.write

Default: false
Runtime: yes
如果设置为true，将不允许写操作。

blocks.metadata

Default: false
Runtime: yes
如果设置为true，alter和drop操作将不允许。这里有一个例外: 仅仅设置一个block.*的设置依然允许alter来更改blocks的设置。

Translog

> 注意
>
> translog提供一个对所有尚未刷新到磁盘的操作的持久化日志。无论何时一行记录插入到一张表(或者更新)的变化将同时被添加到内存缓冲区和translog中。当translog达到一个确切的大小(查看 flush.threshold.size)，或者持有一个确切数量的操作(查看 flush.threshold.ops)，或者达到一个合适的间隔(查看 flush.threshold.period)translog已经同步，刷新到磁盘，并清除。

translog.flush_threshold_ops

Default: unlimited
Runtime: yes
设置translog刷新前的操作数

translog.flush_threshold_size

Default: 200mb
Runtime: yes
设置translog的刷新阈值，默认是200mb。

translog.flush_threshold_period

Default: 30m
Runtime: yes
设置达到flushing的时间间隔，强制落盘。

translog.disable_flush

Default: false Runtime: yes Disable/enable flushing.

关闭或启用flushing

translog.interval

Default: 5s
Runtime: yes
检查是否需要flush的频率,随机在间隔时间和两倍的间隔时间之间。

translog.sync_interval

Default: 5s
Runtime: no
设置index.translog.sync_interval控制translog同步到磁盘后的周期，默认周期是5秒。当设置此间隔，请注在此间隔周期之间的变化将会被记录，不会同步到磁盘，这些变化可能会在记录失败 失败的情况下丢失。

Allocation(分配表分片)

routing.allocation.enable

Default: all Runtime: yes Allowed Values: all | primaries | new_primaries | none 控制对特定表的分片规则

routing.allocation.total_shards_per_node

Default: -1 (unbounded)
Runtime: yes
控制一个节点上允许的总的分片数。

Recovery

recovery.initial_shards

Default: quorum
Runtime: yes
当使用本地网关，一个特殊的分片将会恢复仅当它在集群间的拷贝份数达到法定数。查看recovery.initial_shards可获取更详细的说明。

Warmer(预热)

warmer.enabled

Default: true
Runtime: yes
禁用或启用表预热。表预热允许执行注册的查询语句在表可用之前预热一下表。

Unassigned

unassigned.node_left.delayed_timeout

Default: 1m
Runtime: yes
延迟分配因为一个节点丢失使复制分片变成未分配的分片，默认值是1分钟，为了给节点完全重启的时间。将超时时间设置为0将会立即开始分配。为了增加或减少延迟时间，这个设置可以在运行期间改变。

Column Policy(列规则)

column_policy

Default: dynamic
Runtime: yes
指定表的列规则

Node Specific Settings(节点特定设置)

cluster.name

Default: crate
Runtime: no
crate集群的名称，节点将会按集群的名称加入。

node.name

Runtime: no
节点的名称，如果未设置将会自动生成一个随机的名称。

> 注意
>
> 节点名称在集群里必须是唯一的。

Node Types(节点类型)

Crate支持不同类型的节点。下边的设置区别启动时的节点。

node.master

Default: true
Runtime: no
这个节点是否能被选中当做集群中的主节点。

node.data

Default: true
Runtime: no
该节点是否要存储数据。

node.client

Default: false Runtime: no Shorthand for: node.data=false and node.master=false. 和node.data=false并且node.master=false是等同的。

node.local

Default: false Runtime: no 如果设置为true,节点将会使用本地jvm传输和发现。主要的使用目的是用来测试。

Examples(例子)

一个节点默认作为主节点并且包含数据是合理的。

仅仅包含数据的节点不能成为主节点的节点将主要执行查询的响应。

```
node:
  data: true
  master: false
```

Master-only-nodes这样的节点不包含数据,但是可以作为master,可用来将集群的管理负载从查询负载中分离出来。

```
node:
  data: false
  master: true
```

不包含数据且没有资格作为主节点的节点被称为client-nodes。它们用来分离请求处理负载。

```
node:
  client: true
```

Read-only node(只读节点)

node.sql.read_only

Default: false
Runtime: no

如果设置为true,节点将只允许执行只读操作的sql语句。

Hosts(主机配置)

network.host

Default: 0.0.0.0
Runtime: no
Crate将绑定的ip。这个设置将会同时设置到network.bind_host和network.publish_host这两个配置上。

network.bind_host

Default: 0.0.0.0
Runtime: no
这个设置决定Crate将绑定的地址。只为了仅仅绑定带localhost,将它设置为任何本地的地址或者*local*.

network.publish_host

Runtime: no 这个设置被用于一个Crate节点将自己的地址发布给集群中其他的节点。默认情况下它是第一个非本地地址。

要将crate显式地绑定到特定网卡，请使用下划线之间的网卡地址。例如 *eth0*,这将获取此网卡上的ip地址。使用*eth0:ipv{4,6}*将会显式监听一个ipv6或者ipv4的地址。

Ports(端口)

http.port

Runtime: no
这个配置定义Crate HTTP服务将绑定的TCP的端口范围。默认是4200-4300.通常情况先使用第一个端口范围内未使用端口。

HTTP协议用于所有客户端(java客户端除外)提供REST的服务。

http.publish_port

Runtime: no
客户端使用http端口来和节点通讯。有必要定义这个配置如果绑定的http端口(http.port)不能从外面访问到。例如，在防火墙后面或者Docker容器内部运行它。

transport.tcp.port

Runtime: no
定义Crate的transport服务将绑定的TCP端口范围。默认是4300-4400。通常使用范围内第一个可用的端口。如果这个设置一个整形的值，将被认为显式使用单端口。

传输协议用于内部节点之间通讯。而且java客户端也会使用。

transport.publish_port

Runtime: no
节点将这个端口发布给集群，让集群发现自己。有必要定义这个配置当配置的transport.tcp.port端口不能在外面访问到时，例如运行在防火墙后面或者Docker容器中。

Node Attributes(节点属性)

可以将通用属性应用到节点，使用配置设置如 node.key:value。这些属性可用于自定义分片分配。

参见Awareness设置。

Paths

path.conf

Runtime: no
包含配置文件crate.yml和logging.yml的目录的文件系统路径。

path.data

Runtime: no
Crate节点存储数据的目录(表数据和集群元数据)。

这个配置可以包含逗号分隔路径列表。在这种情况下，Crate将数据分散到各个目录中就像RAID 0一样。

path.work

Runtime: no
处理临时目录的创建和使用在操作期间。这个目录包含许多不应该修补的内部文件。

path.logs

Runtime: no
存储日志文件的目录。可以当做变量在logging.yml文件中使用。

例如： appender: file: file: ${path.logs}/${cluster.name}.log

path.repo

Runtime: no

A list of filesystem or UNC paths where repositories of type fs may be stored.

一个为文件系统的或者UNC (Universal Naming Convention)路径，文件系统类型的仓库将被存储。

没有此项设置，一个Crate用户可以写入快照文件到任意Crate进程可写的目录。为了防止此安全问题，可能的路径必须列在白名单中。

查看 location fs仓库类型的设置。

Plugins(插件)

plugin.mandatory

Runtime: no 一个节点需要启动的一个插件列表。若果没有插件都列在这，Crate节点将会启动失败。

Memory(内存)

bootstrap.mlockall

Runtime: no Default: false
当jvm在进行内存页交换操作时，Crate的性能将会很差:你应该保证Crate从来不进行内存页交换。如果设置为true，Crate将会使用 mlockall系统调用在启动时确保Crate的内存页锁定在内存中。

Garbage Collection(垃圾回收)

当jvm在不同的内存池上垃圾回收时间太长时，Crate将会记录。以下的设置可以被用作调整超时时间。

monitor.jvm.gc.young.warn

Default: 1000ms
Runtime: no
如果超过配置的回收Eden Space (heap)区的时间timespan，Crate将会打印一个warn日志。

monitor.jvm.gc.young.info

Default: 1000ms
Runtime: no
如果超过配置的回收Eden Space (heap)区的时间timespan，Crate将会打印一个info日志。

monitor.jvm.gc.young.debug

Default: 1000ms
Runtime: no
如果超过配置的回收Eden Space (heap)区的时间timespan，Crate将会打印一个debug日志。

monitor.jvm.gc.old.warn

Default: 1000ms
Runtime: no
如果超过配置的回收Old Gen / Tenured Gen (heap)区的时间timespan，Crate将会打印一个warn日志。

monitor.jvm.gc.old.info

Default: 1000ms
Runtime: no
如果超过配置的回收Old Gen / Tenured Gen (heap)区的时间timespan，Crate将会打印一个info日志。

monitor.jvm.gc.old.debug

Default: 1000ms
Runtime: no
如果超过配置的回收Old Gen / Tenured Gen (heap)区的时间timespan，Crate将会打印一个debug日志。

Elasticsearch HTTP REST API(Elasticsearch的HTTP REST服务API)

es.api.enabled

Default: false
Runtime: no
开启或关闭elasticsearch的HTTP REST API。

> 警告
> 通过elasticsearch API操纵你的数据和不通过SQL可能结果有些数据冲突。 你将被警告!

Blobs

blobs.path

Runtime: no
存储分配给此node的blob数据的目录。

默认blob数据存储在和通常数据文件存储目录相同的目录中。一个相对与CRATE_HOME的目录中。

Repositories(仓库)

仓库用来恢复一个Crate集群。

repositories.url.allowed_urls

Runtime: no
这个设置仅仅应用类型为url的仓库。

使用此设置，可以指定一个URL列表，如果创建了类型为url的仓库，则允许使用这些URL。

host，path，query和fragment parts中支持通配符。

这个配置是一个防止强制访问resources的安全措施。

总而言之,支持的协议可严格使用repositories.url.supported_protocols来设置。

repositories.url.supported_protocols

Default: http, https, ftp, file and jar
Runtime: no
由类型为url的仓库支持的协议列表。

jar协议被用作访问jar类型的文件。要看更多的信息，查看 JarURLConnection documentation 文档。

参见path.repo配置。

Cluster Wide Setting(集群围配置)

集群当前使用的所有配置都可以通过查询sys.cluster.setting获得。大部分集群的配置都可以在运行期间通过SET/RESET语句改变 。每个设置都被文档化存储。不建议将集群范围的配置添加到每一个集群节点上的crate.yml配置文件中,如果这样将会使每个节点都有一个不同的集群配置，从而导致集群有不确定性的行为。

Collecting Stats(收集状态信息)

stats.enabled

Default: false
Runtime: yes
是否收集统计集群的信息。

stats.jobs_log_size

Default: 10000
Runtime: yes
sys.jobs_log记录着每个节点上用于性能分析的作业的数量。当达到jobs_log_size的大小后，较早的记录将被删除。一个单独的sq语句将引发集群上一个作业的执行。设置值较大时将会产生很多结果，从而也需要获取较大的内存。设置成0将禁用收集作业信息。

stats.operations_log_size

Default: 10000
Runtime: yes
sys.operations_log表记录着每个节点上用于性能分析的操作次数。当operations_log_size达到上限时较早的记录会被删除。一个作业由多个操作组成。较大的值将产生更多结果，从而也会使用更多的内存。设置为0将禁用收集操作信息。

Usage Data Collector(数据使用情况收集器)

数据使用情况收集器的设置是只读的而且不能在运行期间设置。请查看数据使用情况收集器获取更加详细的使用说明。

udc.enabled

Default: true
Runtime: no
true:开启数使用率的收集器 false:禁用数使用率的收集器

udc.initial_delay

Default: 10m
Runtime: no
启动后第一次ping的延迟。 此字段期望一个long或者double或者一个带时间后缀 (ms, s, m, h, d, w)的字符串。

udc.interval

Default: 24h
Runtime: no 一个UDC ping的间隔。 此字段期望一个long或者double或者一个带时间后缀的字符串。

udc.url

Default: https://udc.crate.io Runtime: no 要ping的URL地址。

Graceful Stop(正常停止)

默认配置下，当Crate进程停止时最简单的方式就是shuts down，可能使一些shards(分片)处于不可用的状态将导致集群处于红色状态而且使许多需要当前不可用shards的查询失败。为了安全地停止Crate节点，可以使用正常停止程序。 以下集群的配置将被用来改变集群节点的停止行为:

cluster.graceful_stop.min_availability

Default: primaries
Runtime: yes
Allowed Values: none | primaries | full
none:无最小可用数据是必须的。节点可以停止即使记录停止后丢失。 primaries:在节点停止后至少所有主分片需要处于可用状态，副本可能丢失。 full: 在节点关闭之后，所有的记录和副本需要处于可用状态。所有的数据都是可用的。

> 注意
> 这个选项将被忽略如果集群中仅仅有一个节点。

cluster.graceful_stop.reallocate

Default: true
Runtime: yes
true:正常停止命令允许在节点停止前重新分配分片，以确保min_availability设置的最小数据可用性。 false:如果集群将需要重新分配分片确保最小可用性min_availability设置的最小数据可用性，则正常停止命令将会失败。

> 注意 确保你有足够的节点和磁盘空间来重新分配。

cluster.graceful_stop.timeout

Default: 2h Runtime: yes
定义正常停止时等待重新分片进程完成的最大时间。强制设置将会使关闭进程进入此超时时间。 此字段期望一个long或者double或者一个带时间后缀(ms, s, m, h, d, w)的字符串。

cluster.graceful_stop.force

Default: false
Runtime: yes
当正常停止时间超过cluster.graceful_stop.timeout设置的超时时间时，强制停止节点。

Bulk Operations(批量操作)

涉及大量含的SQL DML语句例如COPY FROM, INSERT 或 UPDATE，可能消耗很大的资源和时间。以下的设置可以改变这些查询的行为。

bulk.request_timeout

Default: 1m Runtime: yes
内部基于分片的请求在大量行上执行DML SQL语句时的超时时间。

Discovery(发现机制)

discovery.zen.minimum_master_nodes

Default: 1
Runtime: yes
设置确保一个节点可以看到集群范围内可操作的其他N个可作为主节点的合格节点。推荐

> 设置大于1的值当在集群里运行多于两个节点时。
>
> ### discovery.zen.ping_timeout
>
> Default: 3s
> Runtime: yes
> 设置发现其他节点时，等待其他节点ping的响应包的等待时间。当较慢或者阻塞的网络中将此值可设置较大一点可以使发现失败率降到最低。
>
> ### discovery.zen.publish_timeout
>
> Default: 30s
> Runtime: yes
> 等待集群中其他节点都变为发布状态的响应时间。 Unicast Host Discovery(单播主机发现)
>
> ### discovery.zen.ping.multicast.enabled
>
> Default: true
> Runtime: no
> 是否启用多播发现机制。 Crate内置支持几种不同节点发现机制，如何获取节点发现的单播主机。最简单的机制是在配置文件中指定主机列表。
>
> ### discovery.zen.ping.unicast.hosts
>
> Default: not set
> Runtime: no
> 当前有两种其他的发现类型：通过DNS和通过EC2的API。 当一个节点启动时启动这些发现类型之一时，它使用以下列出的指定机制的设置来执行查找。主机和端口将用于生成节点发现的单播主机列表。
> 每当重新选举主节点时，集群中的所有节点也执行相同的查找(请参阅集群元数据)。
>
> ### disovery.type
>
> Default: not set
> Runtime: no
> Allowed Values: srv, ec2
> 参见: Discovery.

Discovery via DNS(通过DNS发现)

Crate内部支持DNS发现机制。将discovery.type设置为srv将开启DNS发现机制。 单播主机的顺序由SRV中国定义的优先级来决定。每个主机的权重和名称都在SRV中记录。例如:

```
_crate._srv.example.com. 3600 IN SRV 2 20 4300 crate1.example.com.

_crate._srv.example.com. 3600 IN SRV 1 10 4300 crate2.example.com.

_crate._srv.example.com. 3600 IN SRV 2 10 4300 crate3.example.com.
```

被发现的节点顺序将如下：

```
crate2.example.com:4300, crate3.example.com:4300, crate1.example.com:4300
```

discovery.srv.query

Runtime: no

The DNS query that is used to look up SRV records, usually in the format _service._protocol.fqdn If not set, the service discovery will not be able to look up any SRV records.

用于查找SRV记录的DNS查询(通常格式如 _service._protocol.fqdn),如果不设置，服务器发现将无法查找任何SRV记录。

discovery.srv.resolver

Runtime: no
用于解析DNS记录的DNS服务器的主机名或者IP，如果未设置，或者指定不可解析的主机名或IP，则使用默认解析器。可以使用hostname:port指定自定义端口。

Discovery via EC2(通过EC2发现) Crate内部支持通过EC2 API来发现。可以将discovery.type设置为ec2来开启EC2发现机制。

cloud.aws.access_key

Runtime: no
用于标识API调用的访问关键ID。

cloud.aws.secret_key

Runtime: no 标识API调用的密钥。

> 注意 AWS证书可以由环境变量变量 AWS_ACCESS_KEY_ID和AWS_SECRET_KEY或者通过系统配置属性aws.accessKeyId和aws.secretKey来配置。

Following settings control the discovery(以下的配置控制发现):

discovery.ec2.groups

Runtime: no
一个安全群组列表;id或者名字。仅仅给定群组的实例将被用于单播主机的发现。

discovery.ec2.any_group

Runtime: no
Default: true
Defines whether all (false) or just any (true) security group must be present for the instance to be used for discovery. 定义所有被用于发现的实例是否必须在所有或任何安全组。

discovery.ec2.host_type

Runtime: no
Default: private_ip
允许的值: private_ip, public_ip, private_dns, public_dns

Defines via which host type to communicate with other instances. 定义通过哪种主机类型和同其他实例通信。

discovery.ec2.availability_zones

Runtime: no
可用区域列表。只有指定可用区域内的指定实例才会用于单播主机发现。

discovery.ec2.ping_timeout

Runtime: no
Default: 3s
在发现期间对已存在的EC2实例的进行ping的超时时间。如果没有时间后缀的情况下，默认使用毫秒。

discovery.ec2.tag.

Runtime: no 可使用discovery.ec2.tag.前缀加上标签名来过滤发现哪些EC2实例。例如要过滤值为dev的环境变量的实例，您的设置如下: discovery.ec2.tag.environment: dev.

cloud.aws.ec2.endpoint

Runtime: no
If you have your own compatible implementation of the EC2 API service you can set the enpoint that should be used.

若果你有自己对EC2服务API服务可兼容的一个实现，你可以设置为此服务。

Routing Allocation(路由分配)

cluster.routing.allocation.enable

Default: all
Runtime: yes
允许的值: all | none | primaries | new_primaries all:允许所有的分片分配，集群可能分配所有类型的shards(分片)。 none:不允许分配分片。将没有分片被移除或者创建。 primaries:只有主分片可以被移除或者创建。这个将包括已存在的主分片。

new_primaries allows allocations for new primary shards only. This means that for example a newly added node will not allocate any replicas. However it is still possible to allocate new primary shards for new indices. Whenever you want to perform a zero downtime upgrade of your cluster you need to set this value before gracefully stopping the first node and reset it to all after starting the last updated node.

new_primaries:仅允许为新的主分片分配。这意味着新添加的节点将不会分配任何副本。然而任然有可能分配主分片给新的索引。无论何时要对集群执行零停机升级，需要先设置此值，然后正常停止第一个节点并在启动最后更新的节点后将其重置为all。

> 注意

此分配设置对主分片的恢复没有任何作用,即使 cluster.routing.allocation.enable被设置为none,节点重启后节点依然会立即恢复它们本地未分配的主分片，以防满足recovery.initial_shards的设置。

cluster.routing.allocation.allow_rebalance

Default: indices_all_active
Runtime: yes
允许的值: always | indices_primary_active | indices_all_active 允许根据集群中所有索引分片

的总状态控制何时进行重新平衡。默认设置为indices_all_active以减少初始恢复期间的抖动。

cluster.routing.allocation.cluster_concurrent_rebalance

Default: 2
Runtime: yes
定义集群允许范围内并发重新平衡任务的任务数。

cluster.routing.allocation.node_initial_primaries_recoveries

Default: 4
Runtime: yes

定义每个节允许的初始恢复数。由于大多时候使用本地网关，这些应该很快而且我们能处理更多的节点而不用创建新的负载。

cluster.routing.allocation.node_concurrent_recoveries

Default: 2 Runtime: yes 一个节点上允许多少个并发恢复任务。

Awareness(感知)

集群分配感知允许在通用属性相关的节点间配置分片和副本。

cluster.routing.allocation.awareness.attributes

Runtime: no 定义用于感知一个分片和它的副本的节点属性。例如，我们定义一个属性rack_id，启动两个节点将其rack_id设置为rack_one，然后部署一个单表带有5个分片和一个副本。这个表将完全部署在当前节点上(五个分片每个分片一个副本，总共10个分片)。 现在，如果我们启动两个或更多个节点，将node.rack_id设置为rack_two,分片将重新定位甚至跨节点的分片，但是一个分片和它的副本将不会分配同样的rack_id值。 感知属性可设置多个值。

cluster.routing.allocation.awareness.force.*.values

Runtime: no
属性所在的分片将被强制分配。*是一个感知属性的通配符，可以通过cluster.routing.allocation.awareness.attributes来设置。假如我们来配置一个感知属性和值

zone1,zone2,启动两个node.zone配置都为zone1的节点，并且创建一个5个分片一个副本的表。表将被创建，但仅仅有5个分片被分配。只有当我们启动多个分片而且node.zone设置为zone2，副本分片才会将被分配。

Balanced Shards(平衡分片)

所有这些值都是相对的。前三个用于将三个独立的加权函数组成一个。当没有允许的行为使得每个节点的权重靠近在一起超过第四设值的时候，集群是平衡的。可能不允许执行一些操作如强制感知或者分配过滤。

cluster.routing.allocation.balance.shard

Default: 0.45f
Runtime: yes
为分片在一个节点上分配定义权重因数。提高这个值提高了均衡集群中所有节点上的分片的数量的趋势。

cluster.routing.allocation.balance.index

Default: 0.5f Runtime: yes

Defines a factor to the number of shards per index allocated on a specific node (float). Increasing this value raises the tendency to equalize the number of shards per index across all nodes in the cluster.

定义在特定节点(float)上分配的每个索引的分片数量的因子。增大这个值将提高均衡集群中所有节点上每个索引分片的数量的趋势。

cluster.routing.allocation.balance.primary

Default: 0.05f
Runtime: yes
Defines a weight factor for the number of primaries of a specific index allocated on a node (float). Increasing this value raises the tendency to equalize the number of primary shards across all nodes in the cluster.

定义在节点(float)上分配的特定索引的基数的权重因子。增大此值将提高平衡集群中主分片数量的趋势。

cluster.routing.allocation.balance.threshold

Default: 1.0f
Runtime: yes
Minimal optimization value of operations that should be performed (non negative float). Increasing this value will cause the cluster to be less aggressive about optimising the shard balance.
应该执行的操作的最小优化值(非负浮点数)。增加此值将降低集群在优化分片平衡方面的积极性。

Cluster-Wide Allocation Filtering(集群范围分配过滤)

Allow to control the allocation of all shards based on include/exclude filters. E.g. this could be used to allocate all the new shards on the nodes with specific IP addresses or custom attributes. 允许所有分片的分配包括过滤器包含或排除过的分片。例如，这可以用于在特有的ip地址或自定义属性的节点上分配所有新分片。

cluster.routing.allocation.include.*

Runtime: no
Place new shards only on nodes where one of the specified values matches the attribute. e.g.: cluster.routing.allocation.include.zone: “zone1,zone2”
仅在节点上分配新的分片，其中指定的值与改属性匹配。例如：
cluster.routing.allocation.include.zone: “zone1,zone2”

cluster.routing.allocation.exclude.*

Runtime: no
当没有特殊的值可以匹配到将只放置新的分片在节点上，例如：
cluster.routing.allocation.exclude.zone: “zone1”

cluster.routing.allocation.require.*

Runtime: no
用于指定一些规则，所有规则必须匹配到一个节点，以便为其分配一个分片。这与包括其中将任何规则匹配的节点的情况相反。

Disk-based Shard Allocation(基于磁盘的分片分配)

cluster.routing.allocation.disk.threshold_enabled
Default: true
Runtime: yes
根据磁盘使用情况阻止节点上的分片分配。

cluster.routing.allocation.disk.watermark.low

Default: 85%
Runtime: yes
定义分片分配的磁盘最小阈值。将不会在磁盘使用量大于此值的节点上分配新的分片。可以设置为绝对字节值(例如500mb),防止在集群中比该值小的磁盘空间的节点上分配新的分片。

cluster.routing.allocation.disk.watermark.high

Default: 90%
Runtime: yes
设置分片分配的最高磁盘阈值限制。如果节点上的磁盘使用率超过此值，集群将尝试将现有分片重新定位到另一个节点。它可以设置为绝对字节值(例如500mb),以便从具有比该值小的可用磁盘空间的节点重新定位分片。 默认情况下，集群将每隔30秒检查有关节点的磁盘使用情况信息。这也可以通过设置cluster.info.update.interval设置来更改。

Recovery(恢复)

indices.recovery.concurrent_streams

Default: 3
Runtime: yes
设置当一个分片从对等点恢复时能够打开的并发流的上限。

indices.recovery.file_chunk_size

Default: 512kb
Runtime: yes
从源分片拷贝分片数据的特殊块大小。如果将indices.recovery.compress设置为true将使用压缩。

indices.recovery.translog_ops

Default: 1000
Runtime: yes
指定在恢复过程中单个请求在分片之间传输的事务日志行数。如果首先达到 indices.recovery.translog_size 设置的值，这个请求将忽略这个值。

indices.recovery.translog_size

Default: 512kb
Runtime: yes
指定在恢复过程中单个请求在分片之间传输的事务日志数据量。如果首先达到 indices.recovery.translog_size 设置的值，这个请求将忽略这个值。

indices.recovery.compress

Default: true
Runtime: yes
设置在恢复过程中是否应该压缩传输的数据。将其设置为false可以降低cpu的压力同时允许更多的数据在网路间传输。

indices.recovery.max_bytes_per_sec

Default: 40mb
Runtime: yes
指定分片在恢复期间每秒最大传输的字节数。可以设置为0来禁用限制。与 indexes.recovery.concurrent_streams类似，此设置允许控制恢复期间的网络使用情况。 更高的值可能导致更高的网络利用率，但也可以加快的恢复过程。

indices.recovery.retry_delay_state_sync

Default: 500ms Runtime: yes
设置在尝试恢复之前由于集群状态同步而导致问题后的等待的时间。

indices.recovery.retry_delay_network

Default: 5s Runtime: yes 设置在尝试恢复前由于网络原因导致问题后的等待时间。

indices.recovery.retry_activity_timeout

Default: 15m
Runtime: yes
设置空闲恢复失败的时间间隔。

indices.recovery.retry_internal_action_timeout

Default: 15m
Runtime: yes
设置作为恢复一部分的内部请求的超时时间。

indices.recovery.retry_internal_long_action_timeout

Default: 30m
Runtime: yes
定义作为恢复的一部分内部请求的超时时间，这些请求预计需要很长时间。默认两次retry_internal_action_timeout时间。

Store Level Throttling

indices.store.throttle.type

Default: merge
Runtime: yes
Allowed Values: all | merge | none
允许对存储模块的合并（或所有）进程进行限制。

indices.store.throttle.max_bytes_per_sec

Default: 20mb
Runtime: yes
如果通过indices.store.throttle.type启用限制，这个设置指定一个存储模块每秒操纵的最大字节数。

Query Circuit Breaker

查询请求将跟踪内存使用情况在一个查询执行期间。如果一个请求消费太多的内存或者集群已经接近内存使用上限，请求将被中断以维持集群正常运行。

indices.breaker.query.limit

Default: 60%
Runtime: yes
指定查询中断的限制。提供的值可能使一个绝对的值，字节大小(例如 1mb)或者堆的百分比

(例如 12%)。值-1可以禁用查询中断，同时任会记录内存使用情况。

indices.breaker.query.overhead

Default: 1.09
Runtime: no
所有数据估算都要相乘的常数值，以确定最终额估算。

Field Data Circuit Breaker(数据断路器)

估算将数据加载到内存所需要的堆存储大小。如果到达确切的限制，将会引发一个异常。

indices.fielddata.breaker.limit

Default: 60%
Runtime: yes
指定fielddata断路器的jvm的堆的限制

indices.fielddata.breaker.overhead

Default: 1.03
Runtime: yes
所有现场数估算都要乘以的常数用来决定最终得估算值。

Request Circuit Breaker

请求断路器允许每个请求估算的堆内存的大小。如果单个请求超过指定的内存量，则会引发异常。

indices.breaker.request.limit

Default: 40%
Runtime: yes
为请求断路器指定jvm的堆内存限制。

indices.breaker.request.overhead

Default: 1.0
Runtime: yes
所有请求估算都要乘以的常数，用来决定最终得估算值。

Threadpools(线程池)

每个节点持有的多个线程池用来改进在每个节点内管理节点。有一些线程池，但是最为重要的包括如下: index: For index/delete operations, defaults to fixed search: For count/search operations, defaults to fixed get: For queries that are optimized to do a direct lookup by primary key, defaults to fixed bulk: For bulk operations, defaults to fixed
refresh: For refresh operations, defaults to cache

index: 索引或删除的操作，默认为fixed
search:计数或查询的操作，默认为fixed
get:通过直接查询一些主要关键字优化查询，默认为fixed
bulk:批量操作，默认为fixed
refresh:刷新操作，默认为cache

threadpool..type

Runtime: no
Allowed Values: fixed | cache
固定持有一个固定数量的线程来处理请求。如果没有可用的线程，它有待处理请求的队列。缓存将会挂起一个线程如果有一个待处理请求的队列。

Fixed Threadpool Settings(固定线程池设置)

如果现线程池的类型被设置为固定大小的将有一些设置项。

threadpool..size

Default index:
Default search: * 3
Default get:
Default bulk:
Runtime: no
线程数

threadpool..queue_size

Default index: 200
Default search: 1000
Default get: 1000
Default bulk: 50
Runtime: no
待处理请求队列的大小。设置为-1将会不限制其大小。

Metadata(元数据)

cluster.info.update.interval

Default: 30s
Runtime: yes
定义集群收集元数据信息的频率，如果没有具体的事件被触发。

Metadata Gateway(元数据网关)

每次元数据发生变化，网关都会将集群的元数据信息持久化到磁盘上。这些数据持久存储在整个集群中，并在节点重新启动后恢复。

gateway.expected_nodes

Default: -1
Runtime: no
这个gateway.expected_nodes设置定义集群状态立即恢复需要等待的节点的数量。这个值应该与集群中节点的数量相等。因为你只想在所有节点都启动后集群状态恢复。

gateway.recover_after_time

Default: 5m
Runtime: no
gateway.recover_after_time设置定义了开始恢复集群状态之前的等待时间，一旦gateway.recover_after_nodes定义的节点数都启动后。如果gateway.recover_after_nodes设置小于gateway.expected_nodes则这个值，则这个值是相关的。

gateway.recover_after_nodes

Default: -1
Runtime: no
gateway.recover_after_nodes定义了需要在集群状态开始恢复之前必须启动的节点数量。理想情况下这个值应该与集群的节点数相等，因为你只需要在所有节点都启动后，恢复集群状态。然而这个值必须大于一半以上数量的集群集群节点数。

Logging(日志)

Crate开箱即用，使用Log4j 1.2.x系列。它尝试使用YAML简化log4j的配置。日志配置文件在config/logging.yml。 yaml文件准备一组日志配置属性使用PropertyConfigurator来配置，但没有使用log4j繁琐的前缀。这里是一个正在工作总的日志配置的小例子。

```
rootLogger: INFO, console

logger:
# log action execution errors for easier debugging
action: DEBUG


appender:
   console:
     type: console
   layout:
     type: consolePattern conversionPattern: "[%d{ISO8601}][%-5p][%-25c] %m%n"
```

And here is a snippet of the generated properties ready for use with log4j. You get the point.
这是一个属性配置片段，使用log4j来配置。

```
log4j.rootLogger=INFO, console log4j.logger.action=DEBUG
log4j.appender.console=org.elasticsearch.common.logging.log4j.ConsoleAppender
log4j.appender.console.layout=org.apache.log4j.PatternLayout
log4j.appender.console.layout.conversionPattern=[%d{ISO8601}][%-5p][%-25c] %m%n ...
```

Logger Settings(日志设置) 可以在运行期间改变日志的级别。这在调试问题或者不用重启的节点而改变日志级别的情况下特别有用。日志设置是集群范围的而且覆盖了节点定义在logging.yaml里面的日志配置信息。

reset的语句也是支持的，然而仅仅有限制的是，reset使日志配置覆盖只有在集群重启后才会生效。

可以使用以下例子来设置日志级别:

```
SET GLOBAL TRANSIENT "logger.action" = 'INFO';
```

日志设置由日志前缀和变量后缀组成，变量后缀定义应用日志级别的日志记录器的迷名称。

除了分层来命名日志记录器，你可以使用_root前缀来改变日志级别。

在上面的例子中，日志级别INFO用于日志记录器操作。

可能日志级别和Log4j的一样:TRACE, DEBUG, INFO, WARN, and ERROR。必须使用set语句设置一个字符串的值。

> 注意
> 小心使用TRACE日志级别，因为它非常详细，可能会掩盖重要的日志信息，甚至会在在某些情况下填满整个磁盘。

可以通过查询sys.cluster来查看集群当前日志覆盖情况。(查看集群设置)

Environment Variables(环境变量)

CRATE_HOME

指定安装的home目录，可以用来查找例如配置文件config/crate.yml或者数据目录。这个变量通常定义在分发的启动脚本里。在大多数情况下是包含bin/crate可执行文件的父目录。

CRATE_HOME:Crate安装的home目录。用于引用默认的配置文件，数据路径，日志文件等。所有配置的相对路径将使用此目录作为目录。

CRATE_HEAP_SIZE

这个变量指定了JVM可以使用的内存大小。

这个环境变量可以使用g或者m的前缀，例如:

```
CRATE_HEAP_SIZE=4g
```

在Crate中确切的操作需要在一个时间在内存中处理大量的行。如果JVM可以分配的堆的数量太少，这些操作将会以一个OutOfMemery的异常失败。

所以选择一个足够高的值最为预期使用非常重要。但是有两个限制:

最大使用内存的50%。

注意除了Crate的堆之外还有另一个内存使用者：我们的数据存储引擎。它通过底层操作系统缓存一些设计的内存数据结构。Lucene索引被分割成一些段文件，每一个文件是不可变的而且将不会更改。这使得他们是有超级缓存友好而且底层操作系统将会将热的端驻留在内存中使访问更快。所以如果所有的系统内存都分配给Crate的堆，Lucene将会没有任何剩余，从而可能引起严重的性能问题。

> 注意

好的建议是给Crate HEAP分配50%可用的内存同时留下50%的剩余。他将不会空闲，Lucene将会使用剩余的内存。

切勿使用超过**30.5**千兆字节

为了在x64位系统上节约内存，java虚拟机Hotspot使用一个叫做压缩普通对象指针(oops)的技术。

在堆中有指向java对象的指针只消耗32位，这将会节省很多空间。通过32位的值乘以因子8计算本地64位指针的值，而且将它添加到基础的堆地址。这允许jVM寻址32GB的堆。

如果你配置的堆大于32GB则压缩指针将不会使用。实际上在堆中将有更少的空间，因为对象指针现在消耗两倍的大小。

此边界将是所有JVM应用程序的堆大小的上限。

> 注意

为了确保使用压缩指针不管是否Crate是否运行，建议配置的堆大小小于或者等于30.5GB(30500m),像一些JVM仅仅支持压缩指针到那个值。

Running Crate on machines with huge RAM(在大内存的机器上运行Crate)

如果机器上有更多可用的物理内存，建议运行多于一个的Crate实例在那台机器上，每一个实例使用大概30.5GB对大小。但是任然要给lucene留有更多的内存大小。

在这种情况下考虑在你的配置中添加如下配置：

cluster.routing.allocation.same_shard.host: true

如果在一台机器上运行多个实例，这将会阻止将一个分片的主分片和副本分片分配在一台机器上。

CRATE CLI

下载并解压tar.gz的安装包，你将会调用bin下面可运行的crate。可执行文件叫crate。

最简单的启动Crate实例的方式是不带参数调用crate。这将会在前台启动进程。

```
sh$ ./bin/crate
```

我们也可以在使用-d参数在后台启动Crate。当在后台启动Crate时，将进程id写到一个pid文件很有帮助，这样你可以轻松地找到进程ID:

```
sh$ ./bin/crate -d -p ./crate.pid
```

要停止在后台运行的进程必须向其发送TERM或者INT的信号

```
sh$ kill -TERM `cat ./crate.pid`
```

可执行的Crate支持以下命令行的选项:

Command Line Options(命令行选项)

```
-d  在后台启动守护进程
-h  打印使用率详情
-p  <pidfile>记录pid到一个文件
-v  打印版本信息
-D  设置一个java系统属性值
-X  设置一个非标准的java选项
```

例如:

```
sh$ ./bin/crate -d -p ./crate.pid
```

Signal Handling(信号处理)

The Crate process can handle the following signals. Crate进程可以控制如下的信号。

Signal	Description
TERM	停止一个正在运行的Crate进程 kill -TERM `cat/path/to/pidfile.pid`
INT	停止一个正在运行的Crate进程 和TERM一样.
USR2	正常停止一个正在运行的Crate. 查看零宕机升级获取更多信息 kill -USR2 `cat /path/to/pidfile.pid` windwos上不支持USR2.

HELLO CRATE

让我们开始快速了解Crate怎么使用。本教程使用Crate发行版附带的crash命令行SQL shell。

```
sh$ ./bin/crash
```

首先连接到一个运行着的Crate:

```
cr> \connect 127.0.0.1:4200;
+-----------------------+-----------+---------+-----------+---------+
| server_url            | node_name | version | connected | message |
+-----------------------+-----------+---------+-----------+---------+
| http://127.0.0.1:...  | crate     | ...     | TRUE      | OK      |
+-----------------------+-----------+---------+-----------+---------+
CONNECT OK
...
```

在这个指导中，我们为Twitter tweets创建一个数据库。让我们来创建一个tweets数据表，表中有我们的需要的数据列：

```
cr> create table tweets ( ... created_at timestamp,
... id string primary key,
... retweeted boolean,
... source string INDEX using fulltext,
... text string INDEX using fulltext,
... user_id string
... ); CREATE OK, 1 row affected (... sec)
```

现在我们准备插入第一个tweet:

```
cr> insert into tweets
... values (1394182937, '1', true, 'web', 'Don''t panic', 'Douglas');
INSERT OK, 1 row affected (... sec)
```

And another: 和另一个:

```
cr> insert into tweets
... values (
... 1394182938,
... '2',
... true,
... 'web',
... 'Time is an illusion. Lunchtime doubly so',
... 'Ford'
... );
INSERT OK, 1 row affected (... sec)
```

为了查询插入的tweets可以使用SELECT语句。这里使用一个过滤条件仅仅查看Ford的tweets信息：

```
cr> select * from tweets where id = '2';
+------------+----+-----------+--------+-------------------------...-+---------+
| created_at | id | retweeted | source | text                        | user_id |
+------------+----+-----------+--------+-------------------------...-+---------+
| 1394182938 | 2  | TRUE      | web    | Time is an illusion. Lun... | Ford    |
+------------+----+-----------+--------+-------------------------...-+---------+
SELECT 1 row in set (... sec)
```

> 参见
>
> 数据定义 - 查看crate建表语句支持的哪些选项，并学习更多的分片和复制的详细信息。
>
> 数据操纵 - 为了学习怎么导入,导出，插入，更新或者删除记录。
>
> Crate查询 - 过滤，排序，分组和强大的全文查询。学习怎么查找你的数据。
>
> 配置 - 在你适用过Crate后并准备在生产环境使用前，请你应该查看一下该章节!

数据定义

表基础

使用CRATE TABLE命令创建一个表。你必须至少为一个表指定一个名称和它的列名。查看数据类型来获取支持的数据类型的更详细的信息。

我们来创建一个简单的带两个列的的表，其中一个列为integer，一个为string:

```
cr> create table my_table (
...   first_column integer,
...   second_column string
... );
CREATE OK, 1 row affected (... sec)
```

可以使用DROP TABLE的命令来删除一个表:

```
cr> drop table my_table;
DROP OK, 1 row affected (... sec)
```

DROP TABLE命令有一个可选择的语句IF EXISTS，这将会防止生成一个错误如果指定的表不存在:

```
cr> drop table if exists my_table;
DROP OK, 0 rows affected (... sec)
```

Schemas(模式)

表可以在不同的schema中创建。这是在创建表时隐式创建的而且不会显式创建。如果一个schema不存在，它将会被创建：

```
cr> create table my_schema.my_table (
...   pk int primary key,
...   label string,
...   position geo_point
... );
CREATE OK, 1 row affected (... sec)
```

```
cr> select table_schema, table_name from information_schema.tables
... where table_name='my_table';
+--------------+------------+
| table_schema | table_name |
+--------------+------------+
| my_schema    | my_table   |
+--------------+------------+
SELECT 1 row in set (... sec)
```

以下的schema的名字是被限制的而且不能被使用:

- blob
- information_schema
- sys

> 注意
>
> Schema是表基本的命名空间。没有访问控制。每个人都能查看和操作每个schema中的表。

只要存在有相同schema的表，用户创建的schema就存在。若果那个schema的最后一张表被删除，这个schema也就不存在了。

```
cr> drop table my_schema.my_table ;
DROP OK, 1 row affected (... sec)
```

每一个表创建的时候没有一个隐式的schema的名字，将会创建在doc这个schema中:

```
cr> create table my_doc_table (
...   a_column byte,
...   another_one geo_point
... );
CREATE OK, 1 row affected (... sec)
```

```
cr> select table_schema, table_name from information_schema.tables
... where table_name='my_doc_table';
+--------------+--------------+
| table_schema | table_name   |
+--------------+--------------+
| doc          | my_doc_table |
+--------------+--------------+
SELECT 1 row in set (... sec)
```

Naming restrictions(命名限制)

表,schema和列标识将不会和保留关键字有相同的名字。请查看Lexical Structure这一章获取关于命名的详细信息。

总之,表和schema的名称的字符和长度被严格限制。如下:

- 不能包含如下的字符: \ / * ? " < > | , #
- 不能包含大写的字母
- 不能以下划线开头: _
- 当使用utf-8编码是不能超过255个字节大小(此限制适用于可选的模式限制的表名)

列名称受字符方面的限制。如下:

- 不能包含以下字符: ['] .
- 不能以下划线开头: _

DATA MANIPULATION(数据操作)

Inserting Data(插入数据)

使用SQL语句向Crate插入数据。

> 注意
> 列总是按照列名称的字母顺序排序。如果省略插入列，则VALUES子句中的必须和表中的列按顺序对应。

插入一行：

```
cr> insert into locations (id, date, description, kind, name, position)
... values (
...   '14',
...   '2013-09-12T21:43:59.000Z',
...   'Blagulon Kappa is the planet to which the police are native.',
...   'Planet',
...   'Blagulon Kappa',
...   7
... );
INSERT OK, 1 row affected (... sec)
```

当插入单行数据时如果有错误发生，将返回错误信息。

可以为INSERT语句定义多个values，这样一次(aka. bulk insert)就可以插入多行数据:

```
cr> insert into locations (id, date, description, kind, name, position) values
... (
...   '16',
...   '2013-09-14T21:43:59.000Z',
...   'Blagulon Kappa II is the planet to which the police are native.',
...   'Planet',
...   'Blagulon Kappa II',
...   19
... ),
... (
...   '17',
...   '2013-09-13T16:43:59.000Z',
...   'Brontitall is a planet with a warm, rich atmosphere and no mountains.',
...   'Planet',
...   'Brontitall',
...   10
... );
INSERT OK, 2 rows affected (... sec)
```

当一次插入多行时，如果某些行出错，将不会返回错误信息信息，但插入的行数将会根据失败的行数而减少。

当向包含生成列的表中插入数据时，生成列的值将被安全地忽略。它是在插入时生成的：

```
cr> insert into debit_card (owner, num_part1, num_part2) values
... ('Zaphod Beeblebrox', 1234, 5678);
INSERT OK, 1 row affected (... sec)
```

如果给定一个值，则会根据列生成子句对当前要插入的行进行校验:

```
cr> insert into debit_card (owner, num_part1, num_part2, check_sum) values
... ('Arthur Dent', 9876, 5432, 642935);
SQLActionException[SQLParseException: Given value 642935 for generated column does not
 match defined generated expression value 642936]
```

按查询插入数据(Inserting Data By Query)

可以使用query语句替换values语句来插入数据。源表和目的表中列的数据类型可以不同，只要数据类型可以转换。这样为以下的操作提供了可能，重建表数据，重命名一个字段，改变一个字段的数据类型或者将一个普通表转换成一个分区表。

改变一个字段的数据类型，在这种情况下，将字段position的数据类型从integer改为short:

```
cr> create table locations2 (
...     id string primary key,
...     name string,
...     date timestamp,
...     kind string,
...     position short,
...     description string
... ) clustered by (id) into 2 shards with (number_of_replicas = 0);
CREATE OK, 1 row affected (... sec)
```

```
cr> insert into locations2 (id, name, date, kind, postition, description)
... (
...     select id, name, date, kind, position, description
...     from locations
...     where position < 10
... );
INSERT OK, 14 rows affected (... sec)
```

在locations表之外创建一个新以year分区的分区表：

```
cr> create table locations_parted (
...     id string primary key,
...     name string,
...     year string primary key,
...     date timestamp,
...     kind string,
...     position integer
... ) clustered by (id) into 2 shards
... partitioned by (year) with (number_of_replicas = 0);
CREATE OK, 1 row affected (... sec)
```

```
cr> insert into locations_parted (id, name, year, date, kind, postition)
... (
...     select
...         id,
...         name,
...         date_format('%Y', date),
...         date,
...         kind,
...         position
...     from locations
... );
INSERT OK, 16 rows affected (... sec)
```

最后通过查询插入将数据插入到分区表:

```
cr> select table_name, partition_ident, values, number_of_shards, number_of_replicas
... from information_schema.table_partitions
... where table_name = 'locations_parted'
... order by partition_ident;
+-----------------+-----------------+-----------------+-----------------+----------
----------+
| table_name      | partition_ident | values          | number_of_shards | number_of
_replicas |
+-----------------+-----------------+-----------------+-----------------+----------
----------+
| locations_parted | 042j2e9n74     | {"year": "1979"} |                2 |
        0 |
| locations_parted | 042j4c1h6c     | {"year": "2013"} |                2 |
        0 |
+-----------------+-----------------+-----------------+-----------------+----------
----------+
SELECT 2 rows in set (... sec)
```

> 注意
>
> limit,offset和order by在内部插叙语句中不支持。

多关键字更新(On Duplicate Key Update)

ON DUPLICATE KEY UPDATE子句用在更新已存在的行，如果由于duplicate-key和现有文档PRIMARY KEY键冲突无法插入时将更新现有行:

```
cr> select
...     name,
...     visits,
...     extract(year from last_visit) as last_visit
... from uservisits order by name;
+----------+--------+------------+
| name     | visits | last_visit |
+----------+--------+------------+
| Ford     |      1 | 2013       |
| Trillian |      3 | 2013       |
+----------+--------+------------+
SELECT 2 rows in set (... sec)
```

```
cr> insert into uservisits (id, name, visits, last_visit) values
... (
...     0,
...     'Ford',
...     1,
...     '2015-09-12'
... ) on duplicate key update
...     visits = visits + 1,
...     last_visit = '2015-01-12';
INSERT OK, 1 row affected (... sec)
```

```
cr> select
...     name,
...     visits,
...     extract(year from last_visit) as last_visit
... from uservisits where id = 0;
+------+--------+------------+
| name | visits | last_visit |
+------+--------+------------+
| Ford |      2 | 2015       |
+------+--------+------------+
SELECT 1 row in set (... sec)
```

可以使用VALUES(column_ident)函数引用没有重复键冲突时插入的值。这个函数特别用于多行插入，可引用当前行的值:

```
cr> insert into uservisits (id, name, visits, last_visit) values
... (
...     0,
...     'Ford',
...     2,
...     '2016-01-13'
... ),
... (
...     1,
...     'Trillian',
...     5,
...     '2016-01-15'
... ) on duplicate key update
...     visits = visits + VALUES(visits),
...     last_visit = VALUES(last_visit);
INSERT OK, 2 rows affected (... sec)
```

```
cr> select
...     name,
...     visits,
...     extract(year from last_visit) as last_visit
... from uservisits order by name;
+----------+--------+------------+
| name     | visits | last_visit |
+----------+--------+------------+
| Ford     |      4 | 2016       |
| Trillian |      8 | 2016       |
+----------+--------+------------+
SELECT 2 rows in set (... sec)
```

也可以使用一个query语句代替values:

```
cr> create table uservisits2 (
...   id integer primary key,
...   name string,
...   visits integer,
...   last_visit timestamp
... ) clustered by (id) into 2 shards with (number_of_replicas = 0);
CREATE OK, 1 row affected (... sec)
```

```
cr> insert into uservisits2 (id, name, visits, last_visit)
... (
...     select id, name, visits, last_visit
...     from uservisits
... );
INSERT OK, 2 rows affected (... sec)
```

```
cr> insert into uservisits2 (id, name, visits, last_visit)
... (
...     select id, name, visits, last_visit
...     from uservisits
... )
... on duplicate key update
...     visits = visits + VALUES(visits),
...     last_visit = VALUES(last_visit);
INSERT OK, 2 rows affected (... sec)
```

```
cr> select
...     name,
...     visits,
...     extract(year from last_visit) as last_visit
... from uservisits order by name;
+----------+--------+------------+
| name     | visits | last_visit |
+----------+--------+------------+
| Ford     |      4 | 2016       |
| Trillian |      8 | 2016       |
+----------+--------+------------+
SELECT 2 rows in set (... sec)
```

更新数据(Updating Data)

要更新Crate中的文档，可使用UPDATE的SQL语句:

```
cr> update locations set description = 'Updated description'
... where name = 'Bartledan';
UPDATE OK, 1 row affected (... sec)
Updating nested objects is also supported:
```

```
cr> update locations set race['name'] = 'Human' where name = 'Bartledan';
UPDATE OK, 1 row affected (... sec)
```

也可以在一个表达式中引用一个列，例如像这样自加一个数字:

```
cr> update locations set position = position + 1 where position < 3;
UPDATE OK, 6 rows affected (... sec)
```

> 注意
> 如果同时更新同一个文档时发生一个版本冲突的异常VersionConflictException。Crate有一个重试的逻辑并且尝试自动解决冲突。

删除数据(Deleting Data)

使用DELETE语句删除Crate中的行:

```
cr> delete from locations where position > 3;
DELETE OK, ... rows affected (... sec)
```

导入/导出(Import/Export)

Importing Data

使用SQL语句 COPY FROM可以将数据导入到Crate中。目前只支持JSON数据格式,一行代表一条数据。 Example JSON data: 例如JSON数据: {"id": 1, "quote": "Don't panic"} {"id": 2, "quote": "Ford, you're turning into a penguin. Stop it."}

> 注意
>
> - COPY FROM将不会转换或校验你的数据。请确保它和你的schema相匹配。
> - 生成列的值如果数据中没有将会计算产生，否则他们将会导入但不校验，所以请确保数据的正确性。
> - 此外，数据中的列名称是区分大小写的(就像他们在sql语句中被引用一样)。 为了将数据导入到分区表中，请参见COPY FROM

从文件URI导入(Import From File URI)

一个使用文件URI导入的例子:

```
cr> copy quotes from 'file:///tmp/import_data/quotes.json';
COPY OK, 3 rows affected (... sec)
```

如果所有文件都在一个文件夹应该使用通配符*:

```
cr> copy quotes from '/tmp/import_data/*' with (concurrency = 1, bulk_size = 4);
COPY OK, 3 rows affected (... sec)
```

通配符也可以只匹配确切的文件:

```
cr> copy quotes from '/tmp/import_data/qu*.json';
COPY OK, 3 rows affected (... sec)
```

详情参见COPY FROM。

(数据导出)Exporting Data

可以使用COPY TO语句导出数据。数据导出以分布式方式导出，这意味着每个节点将导出自己的数据。

副本数据将不会导出。因此导出表的每一行只存储一次。

此例子展示如何将给定表数据导出到以表名和分片id命名的gzip压缩文件中:

```
cr> refresh table quotes;
REFRESH OK...
```

```
cr> copy quotes to DIRECTORY '/tmp/' with (compression='gzip');
COPY OK, 3 rows affected ...
```

可以使用可选的where子句条件过滤一些行问不是导出整张表。如果只需要导出部分数据此方式非常有用:

```
cr> copy quotes where match(quote_ft, 'time') to DIRECTORY '/tmp/' with (compression='
gzip');
COPY OK, 2 rows affected ...
```

有关详情请参见COPY TO。

CRATE查询(QUERYING CRATE)

本章提供了使用SQL语句查询文档的概览。有关表的创建和数据的定义请参阅数据定义。

检索数据(RETRIEVING DATA)

使用SQL的SELECT语句从Crate检索数据。SELECT查询语句中包含的列的名称，实际结果行是一个二位数组，行数和持续时间。 A simple select: 一个简单的查询:

```
cr> select name, position from locations order by id limit 2;
+-------------------+----------+
| name              | position |
+-------------------+----------+
| North West Ripple | 1        |
| Arkintoofle Minor | 3        |
+-------------------+----------+
SELECT 2 rows in set (... sec)
```

如果'*'，将返回所有在schema中定义的列：

```
cr> select * from locations order by id limit 1 offset 1;
+----...+---------------...-+----+-------------+--...+--...+----------+-------------
-----...-+
| date   | description      | id | information | kind | name | position | race
         |
+----...+---------------...-+----+-------------+--...+--...+----------+-------------
-----...-+
| 308... | Motivated by ... | 10 |        NULL | P... | A... | 3        | {"descriptio
n": ...} |
+----...+---------------...-+----+-------------+--...+--...+----------+-------------
-----...-+
SELECT 1 row in set (... sec)
```

可以使用as来改变输出的列的名称:

```
cr> select name as n
... from locations
... where name = 'North West Ripple';
+-------------------+
| n                 |
+-------------------+
| North West Ripple |
+-------------------+
SELECT 1 row in set (... sec)
```

FROM子句(FROM Clause)

FROM子句用于SELECT的查询引用。可以是单个表，多个表，表连接(JOIN)或者子查询(Sub Select)。

一个表被schema和表名称引用，并且也可以是表别名。如果表t仅仅被表名引用，Crate假定使用doc.t。使用CREATE TABLE新创建的Schemas必须显示引用。

以下两种查询是等价的:

```
cr> select name, position from locations
... order by name desc nulls last limit 2;
+------------------+----------+
| name             | position |
+------------------+----------+
| Outer Eastern Rim |        2 |
| North West Ripple |        1 |
+------------------+----------+
SELECT 2 rows in set (... sec)
```

```
cr> select doc.locations.name as n, position from doc.locations
... order by name desc nulls last limit 2;
+------------------+----------+
| n                | position |
+------------------+----------+
| Outer Eastern Rim |        2 |
| North West Ripple |        1 |
+------------------+----------+
SELECT 2 rows in set (... sec)
```

为了简洁，表名也可以是别名:

```
cr> select name from doc.locations as l
... where l.name = 'Outer Eastern Rim';
+------------------+
| name             |
+------------------+
| Outer Eastern Rim |
+------------------+
SELECT 1 row in set (... sec)
```

连接(Joins)

> 注意
>
> Crate目前只支持一组有限的JOIN。查看Joins获取最新的状况。

DISTINCT子句(DISTINCT Clause)

如果执行DISTINCT，将只保留唯一行。结果集中所有其他重复的行将被移除:

```
cr> select distinct date from locations order by date;
+---------------+
| date          |
+---------------+
| 308534400000  |
| 1367366400000 |
| 1373932800000 |
+---------------+
SELECT 3 rows in set (... sec)
```

WHERE子句(WHERE Clause)

一个使用等号操作的简单的where子句的例子:

```
cr> select description from locations where id = '1';
+-------------------------------------...-------------------------------------+
| description                                                                  |
+-------------------------------------...-------------------------------------+
| Relative to life on NowWhat, living on... a factor of about seventeen million. |
+-------------------------------------...-------------------------------------+
SELECT 1 row in set (... sec)
```

比较运算符(Comparison Operators)

在where子句中使用的通常比较运算符适用于所有简单数据类型:

```
操作符(Operator)    详情(Description)
<    小于(less than)
>    大于(greater than)
<=    小于等于(less than or equal to)
>=    大于等于(greater than or equal to)
=    等于(equal)
<>    不等于(not equal)
!=    不等于(not equal) - 等价于 <>
like    匹配给定值的一部分内容(matches a part of the given value)
~    正则匹配(regular expression match)
!~    否定式正则匹配(negated regular expression match)
```

对于字符串，基于Lucene TermRangeQuery执行逐一比较:

```
cr> select name from locations where name > 'Argabuthon' order by name;
+-----------------------------------+
| name                              |
+-----------------------------------+
| Arkintoofle Minor                 |
| Bartledan                         |
| Galactic Sector QQ7 Active J Gamma |
| North West Ripple                 |
| Outer Eastern Rim                 |
+-----------------------------------+
SELECT 5 rows in set (... sec)
```

更多的细节请参见Apache Lucene site

数字和日期类型的比较和标准SQL一致。下面的例子使用支持的ISO日期格式中的一种:

```
cr> select date, position from locations where date <= '1979-10-12' and
... position < 3 order by position;
+--------------+----------+
| date         | position |
+--------------+----------+
| 308534400000 | 1        |
| 308534400000 | 2        |
+--------------+----------+
SELECT 2 rows in set (... sec)
```

有关支持的ISO日期格式的详细说明，请参阅joda date_optional_time网站。

对于自定义的日期类型或者对象映射中的定义的日期类型应该使用相应的格式进行比较，否则比较失败。

谓词(Predicates)

谓词是结果为boolean类型的表达式。

虽然值为boolean类型的标准函数可以在谓词出现的地方使用，但谓词不能在任何地方使用，例如.在一个select语句的结果列里。谓词的语意通常不同于函数的语义。

例如 通过名称和参数类型识别函数。一个引用可以和函数的参数值互换。谓词可以具有不同的语意，例如 强制使用列引用。

以下谓词扩展了where子句中可用条件的范围。

谓词	描述
IS NOT NULL	字段不为空而且不能缺失
IS NULL	字段为空或者缺失
MATCH Predicate	在索引列上执行搜索。请参阅全文搜索或者地理位置搜索的使用。

> 注意 不是所有的谓词都可以有效地利用索引。例如“is null”不能使用索引，而且它的执行效率比其他操作符差。

正则表达式

正则表达式匹配值的操作符。

操作符	描述	例子
~	匹配正则表达式，区分大小写	'foo' ~ '*.foo.*'
~*	匹配正则表达式区分大小写	'Foo' ~ '.foo.*'
!~	不匹配的正则表达式，区分大小写	'Foo' !~ '*.foo.*'
!~*	不匹配的正则表达式，区分大小写	'foo' !~ '.bar.*'

~操作符可以通过正则表达式来匹配一个字符串。如果字符串符合正则匹配规则返回true，否则为false，如果字符串为NULL则返回NULL。

要否定匹配，在签名使用“!”。如果字符串与模式不匹配将返回true，否则false。 允许使用所有的Unicode。

如果在正则表达式中使用PCRE,使用java标准库 java.util.regex中的操作符。

如果不在正则表达式中使用PCRE的特性，则运算符使用Lucene中的正则表达式，该表达式针对Lucene的分词优化正则表达式。

Lucene表达式是基本的POSIX扩展的正则表达式，没有字符类觉一些其它扩展，想元字符#匹配空串或者~匹配非空或其它。默认Lucene的扩展是开启的。详情参阅Lucene的文档

> 注意
> 因为使用使用~* 或者 !~*且不区分大小写的匹配隐式使用标准的java库，Lucene表达式将不会在这起作用。

例如:

```
cr> select name from locations where name ~ '([A-Z][a-z0-9]+)+'
... order by name;
+------------+
| name       |
+------------+
| Aldebaran  |
| Algol      |
| Altair     |
| Argabuthon |
| Bartledan  |
+------------+
SELECT 5 rows in set (... sec)
```

```
cr> select 'matches' from sys.cluster where
... 'gcc --std=c99 -Wall source.c' ~ '[A-Za-z0-9]+((-|--)[A-Za-z0-9]+)*( [^ ]+)*';
+-----------+
| 'matches' |
+-----------+
| matches   |
+-----------+
SELECT 1 row in set (... sec)
```

```
cr> select 'no_match' from sys.cluster where 'foobaz' !~ '(foo)?(bar)$';
+------------+
| 'no_match' |
+------------+
| no_match   |
+------------+
SELECT 1 row in set (... sec)
```

LIKE(模糊查找)

Crate支持LIKE操作。like操作可用来查询部分列值匹配的的行。例如获取所有名字以'Ar'开头的位置信息可使用如下查询:

```
cr> select name from locations where name like 'Ar%' order by name asc;
+-------------------+
| name              |
+-------------------+
| Argabuthon        |
| Arkintoofle Minor |
+-------------------+
SELECT 2 rows in set (... sec)
```

以下的通配符操作是有用的:

% 替代0个或多个字符 _ 替代一个字符

通配符可以在任何字符串中使用。例如，像这样一个更复杂的子句:

```
cr> select name from locations where name like '_r%a%' order by name asc;
+------------+
| name       |
+------------+
| Argabuthon |
+------------+
SELECT 1 row in set (... sec)
```

In order so search for the wildcard characters themselves it is possible to escape them using a backslash: 为了搜索本身的字符，可以使用反斜线转义:

```
cr> select description from locations
... where description like '%\%' order by description asc;
+-------------------------+
| description             |
+-------------------------+
| The end of the Galaxy.% |
+-------------------------+
SELECT 1 row in set (... sec)
```

> 注意 使用like子句的插叙会相当慢。特别是以通配符开始的like子句。因为crate在处理这种情况四必须遍历所有的行而且不使用索引。为了更好的性能考虑使用全文索引。

NOT NOT的否定是一个布尔表达式:

```
[ NOT ] 布尔表达式
```

结果类型是布尔值。

expression	result
true	false
false	true
null	true

> 注意
> 这不符合SQL标准而且不遵循三值逻辑。根据标准NOT(NULL)将会返回NULL。

IN

Crate也支持二元运算符IN,将允许你验证左边的操作数在右边的一组表达式中。当右边表达式中的任意值和左边的操作数相等将返回true，否则为false:

```
cr> select name, kind from locations
... where (kind in ('Star System', 'Planet'))  order by name asc;
 +---------------------+-------------+
 | name                | kind        |
 +---------------------+-------------+
 |                     | Planet      |
 | Aldebaran           | Star System |
 | Algol               | Star System |
 | Allosimanius Syneca | Planet      |
 | Alpha Centauri      | Star System |
 | Altair              | Star System |
 | Argabuthon          | Planet      |
 | Arkintoofle Minor   | Planet      |
 | Bartledan           | Planet      |
 +---------------------+-------------+
 SELECT 9 rows in set (... sec)
```

IS NULL

如果表达式评定为NULL则返回TRUE。如果给定一个引用列的字段中包含NULL或者缺失将返回TRUE。

使用此谓词来检查NULL的值就像使用SQL中的三值逻辑，当比较NULL时将总是返回NULL。

expr: Crate支持的数据类型的表达式。

```
cr> select name from locations where race is null order by name;
+-----------------------------------+
| name                              |
+-----------------------------------+
|                                   |
| Aldebaran                         |
| Algol                             |
| Allosimanius Syneca               |
| Alpha Centauri                    |
| Altair                            |
| Argabuthon                        |
| Galactic Sector QQ7 Active J Gamma |
| North West Ripple                 |
| Outer Eastern Rim                 |
| NULL                              |
+-----------------------------------+
SELECT 11 rows in set (... sec)
```

```
cr> select count(*) from locations where name is null;
+----------+
| count(*) |
+----------+
|        1 |
+----------+
SELECT 1 row in set (... sec)
```

IS NOT NULL

Returns TRUE if expr does not evaluate to NULL. Additionally, for column references it returns FALSE if the column does not exist.

如果表达式不评定为NULL则发回TRUE。总之，对引用列如果不存在将返回FALSE。

Use this predicate to check for non-NULL values as SQL's three-valued logic does always return NULL when comparing NULL.

使用此谓词检查非空值像SQL中三值逻辑当比较NULL时总是返回NULL。

expr: expression of one of the supported Data Types supported by crate.

```
cr> select name from locations where race['interests'] is not null;
+-------------------+
| name              |
+-------------------+
| Arkintoofle Minor |
| Bartledan         |
+-------------------+
SELECT 2 rows in set (... sec)
```

```
cr> select count(*) from locations where name is not null;
+----------+
| count(*) |
+----------+
|       12 |
+----------+
SELECT 1 row in set (... sec)
```

ANY (array)

ANY(或者 SOME)操作符允许在数组内的元素上搜索。这允许查询数组中某些元素的行，例如，等于或大于某些表达。

以下的例子将返回任何数组 race['interests']中包含‘netball’的行:

```
cr> select race['name'], race['interests'] from locations
... where 'netball' = ANY(race['interests']);
+----------------+-----------------------------------------+
| race['name']   | race['interests']                       |
+----------------+-----------------------------------------+
| Bartledannians | ["netball", "books with 100.000 words"] |
+----------------+-----------------------------------------+
SELECT 1 row in set (... sec)
```

```
cr> select race['name'], race['interests'] from locations
... where 'books%' LIKE ANY(race['interests']);
+----------------+-----------------------------------------+
| race['name']   | race['interests']                       |
+----------------+-----------------------------------------+
| Bartledannians | ["netball", "books with 100.000 words"] |
+----------------+-----------------------------------------+
SELECT 1 row in set (... sec)
```

It can also be used on array literals: 也可以用于文本数组:

```
cr> select name, race['interests'] from locations
... where name = ANY (['Bartledan', 'Algol'])
... order by name asc;
+-----------+-----------------------------------------+
| name      | race['interests']                       |
+-----------+-----------------------------------------+
| Algol     | NULL                                    |
| Bartledan | ["netball", "books with 100.000 words"] |
+-----------+-----------------------------------------+
SELECT 2 rows in set (... sec)
```

This way it can be used as a shortcut for name = 'Bartledan' OR name = 'Algol' or any other ANY comparison. 这种方法可以用作为诸如 name = 'Bartledan' OR name = 'Algol' 或者其它比较的快捷方式。 Syntax of the ANY Operator: ANY操作符的语法:

expression operator ANY | SOME (array)

ANY操作符允许在数组中的元素中使用操作符。比较必须返回一个boolean结果。

- 如果任何比较都返回true则ANY的结果就为true。
- 如果没有比较但会true或者一个数组不包含任何元素，则ANY的结果为false。
- 如果其中一个表达式或者数组为null，ANY的结果为NULL。如果没有为true的比较且数组中的人任何元素都为null，ANY的结果也为null。

> 注意
> 以下不支持ANY的操作符:
>
> - ‘is null’和‘is not null’操作符
> - 对象数组
> - 对象表达式

Negating ANY

使用ANY时需要注意的一个重要的事情是对ANY操作取反将不会像普通的比较操作一样会取反。

以下的查询可被翻译为race[‘interests’]中至少有一个元素等于‘netball’的所有行:

```
cr> select race['name'], race['interests'] from locations
... where 'netball' = ANY(race['interests']);
+----------------+----------------------------------------+
| race['name']   | race['interests']                      |
+----------------+----------------------------------------+
| Bartledannians | ["netball", "books with 100.000 words"] |
+----------------+----------------------------------------+
SELECT 1 row in set (... sec)
```

以下的查询使用否定运算符!=可能被翻译为race['interests']中至少有一个元素不等于'netball'的所有行。正如你看到的，再这个例子汇总结果是一样的:

```
cr> select race['name'], race['interests'] from locations
... where 'netball' != ANY(race['interests']);
+----------------+----------------------------------------+
| race['name']   | race['interests']                      |
+----------------+----------------------------------------+
| Minories       | ["baseball", "short stories"]          |
| Bartledannians | ["netball", "books with 100.000 words"] |
+----------------+----------------------------------------+
SELECT 2 rows in set (... sec)
```

否定=查询和上面的完全不同。他将被翻译为获取所有race['interests']没有值等于'netball'的行:

```
cr> select race['name'], race['interests'] from locations
... where not 'netball' = ANY(race['interests']) order by race['name'];
+--------------+-------------------------------+
| race['name'] | race['interests']             |
+--------------+-------------------------------+
| Minories     | ["baseball", "short stories"] |
+--------------+-------------------------------+
SELECT 1 row in set (... sec)
```

同样的行为(尽管涉及不同的比较操作)对操作符是成立的。

LIKE and NOT LIKE

所有的其它的比较操作(IS NULL和IS NOT NULL除外)

Limits

不限制的SELECT查询不会打断你的集群如果匹配到的行超过节点的内存的大小，SELECT语句通过默认的10000行来限制。你可以扩大这个限制使用严格的limit子句。但是鼓励您使用LIMIT和OFFSET使用窗口来遍历潜在的大的结果集而不是扩大默认的limit。

当使用PostgreSQL的有线协议，在SELECT语句上将没有严格10000行的限制。

内部/嵌套对象

Crate支持对象数据类型，简单将一个对象存储到一个列中，甚至可以查询和选择此对象的属性。

选择一个对象内部的属性:

```
cr> select name, race['name'] from locations where name = 'Bartledan';
+-----------+----------------+
| name      | race['name']   |
+-----------+----------------+
| Bartledan | Bartledannians |
+-----------+----------------+
SELECT 1 row in set (... sec)
```

插叙一个内部对象的属性:

```
cr> select name, race['name'] from locations
... where race['name'] = 'Bartledannians';
+-----------+----------------+
| name      | race['name']   |
+-----------+----------------+
| Bartledan | Bartledannians |
+-----------+----------------+
SELECT 1 row in set (... sec)
```

插入对象:

```
cr> insert into locations (id, name, position, kind, race)
... values ('DO', 'Dornbirn', 14, 'City', {name='Vorarlberger',
...     description = 'Very nice people with a strange accent',
...     interests = ['mountains', 'cheese', 'enzian']}
... );
INSERT OK, 1 row affected (... sec)
```

对象数组(Object Arrays)

数组在crate中仅仅包含数。对象数组是一个例外。你可以使用下标表达式访问存储内部或嵌套对象的字段，你可以访问对象数据的字段。因为对象数组不是对象，你将不会得到单个字段的值，而是包含那个字段的所有对象，及对象中所有值的一个数组。

例如:

```
c> select name, information['population'] from locations
... where information['population'] is not null
... order by name;
+------------------+---------------------------+
| name             | information['population'] |
+------------------+---------------------------+
| North West Ripple | [12, 42]                  |
| Outer Eastern Rim | [5673745846]              |
+------------------+---------------------------+
SELECT 2 rows in set (... sec)
```

```
cr> select information from locations
... where information['population'] is not null
... order by name;
+-------------------------------------------------...-----------------------+
| information                                                                |
+-------------------------------------------------...-----------------------+
| [{"evolution_level": 4, "population": 12}, {"evolu...": 42, "popul...": 42}] |
| [{"evolution_level": 2, "population": 5673745846}]                         |
+-------------------------------------------------...-----------------------+
SELECT 2 rows in set (... sec)
```

```
cr> insert into locations (id, name, position, kind, information)
... values (
...   'B', 'Berlin', 15, 'City',
...   [{evolution_level=6, population=3600001},
...   {evolution_level=42, population=1}]
... );
INSERT OK, 1 row affected (... sec)
```

```
cr> refresh table locations;
REFRESH OK, 1 row affected (... sec)
```sh
cr> select name from locations where 4 < ANY (information['evolution_level'])
... order by name;
+------------------+
| name             |
+------------------+
| Berlin           |
| North West Ripple |
+------------------+
SELECT 2 rows in set (... sec)
```

选择数组元素(Selecting Array Elements)

数组元素可以使用一个大于或等于1的值直接选中。最大支持的数组下标是2147483648。使用大于实际值的下标将会返回空值NULL。

```
cr> select name, information[1]['population'] as population from locations
... where information['population'] is not null
... order by name;
+------------------+------------+
| name             | population |
+------------------+------------+
| Berlin           |    3600001 |
| North West Ripple |         12 |
| Outer Eastern Rim | 5673745846 |
+------------------+------------+
SELECT 3 rows in set (... sec)
```

> 注意
> 只支持数组下标中的一个数组符号，例如以下将不会工作:
> select information[1][tags][1] from locations;

数据聚合(Data Aggregation)

Crate支持聚合，使用以下列出的在select语句上的聚合函数。

聚合工作在与查询匹配的所有行，或者使用GROUP BY语句的每个不同组中的匹配行上工作。聚合查询语句没有GROUP BY将总是返回一行，因为它们在当做一个组的匹配行上执行聚合操作。
```

> 参见
> 聚合

名称	参数	描述	返回值类型
ARBITRARY	基本类型(除对象外)列的列名	返回参数列中所有值中未定义的值。可以为NULL	列的类型或者NULL，如果匹配行的列为NULL
AVG / MEAN	一个数字类型或者时间戳类型的列名	返回参数列的算术平均值。忽略NULL值	double或者NULL，如果所有匹配行的列值都是NULL
COUNT(*)	星型作为参数或恒定	统计查询匹配的行数	long
COUNT	列名	统计中包含非NULL的给定列的行数。	long
COUNT(DISTINCT col)	列名	统计给定列的buweiNULL的值的数量	long
GEOMETRIC_MEAN	数字类型或者时间戳类型的列名称	计算正数的几何平均值。	double 或者 NULL 如果匹配到的所有行都为NULL或者为负数。
MIN	数字类型，时间戳或者字符串类型的列名。	返回参数列中最小的值，如果是string则意味着按字典中最小的。NULL值忽略。	输入列类型或者NULL,如果匹配到的行的那一列为空
MAX	数字类型，时间戳或者字符串类型的列名。	返回参数列中值最大的，如果是string则意味着按字典中最大的。NULL值忽略。	输入列类型或者NULL,如果匹配到的行的那一列为空
STDDEV	数字类型，时间戳或者字符串类型的列名。	返回参数列中的值的标准偏差。忽略NULL值	double或者NULL,如果匹配到的行的那一列为NULL
SUM	数字或者时间戳类型的列名	返回参数列的求和值。忽略NULL值	double或者NULL,如果匹配到的行的那一列为NULL。
VARIANCE	数字或时间戳类型的列名	返回参数列中值的方差	double 或者 NULL 若果所有值都为NULL或者我们没有取到任何值

> 注意 聚合仅仅应用在普通索引上，默认所有基本类型的列。更多详情，请参见Plain index(default)。

一些例子:

```
cr> select count(*) from locations;
+----------+
| count(*) |
+----------+
| 15       |
+----------+
SELECT 1 row in set (... sec)
```

```
cr> select count(*) from locations where kind = 'Planet';
+----------+
| count(*) |
+----------+
| 5        |
+----------+
SELECT 1 row in set (... sec)
```

```
cr> select count(name), count(*) from locations;
+-------------+----------+
| count(name) | count(*) |
+-------------+----------+
| 14          | 15       |
+-------------+----------+
SELECT 1 row in set (... sec)
```

```
cr> select max(name) from locations;
+-------------------+
| max(name)         |
+-------------------+
| Outer Eastern Rim |
+-------------------+
SELECT 1 row in set (... sec)
```

```
cr> select min(date) from locations;
+--------------+
| min(date)    |
+--------------+
| 308534400000 |
+--------------+
SELECT 1 row in set (... sec)
```

```
cr> select count(*), kind from locations
... group by kind order by kind asc;
+----------+-------------+
| count(*) | kind        |
+----------+-------------+
| 2        | City        |
| 4        | Galaxy      |
| 5        | Planet      |
| 4        | Star System |
+----------+-------------+
SELECT 4 rows in set (... sec)
```

```
cr> select max(position), kind from locations
... group by kind order by max(position) desc;
+---------------+-------------+
| max(position) | kind        |
+---------------+-------------+
| 15            | City        |
| 6             | Galaxy      |
| 5             | Planet      |
| 4             | Star System |
+---------------+-------------+
SELECT 4 rows in set (... sec)
```sh
cr> select min(name), kind from locations
... group by kind order by min(name) asc;
+-----------------------------------+-------------+
| min(name)                         | kind        |
+-----------------------------------+-------------+
|                                   | Planet      |
| Aldebaran                         | Star System |
| Berlin                            | City        |
| Galactic Sector QQ7 Active J Gamma | Galaxy      |
+-----------------------------------+-------------+
SELECT 4 rows in set (... sec)
```

```
cr> select count(*), min(name), kind from locations
... group by kind order by kind;
+----------+-----------------------------------+-------------+
| count(*) | min(name)                         | kind        |
+----------+-----------------------------------+-------------+
| 2        | Berlin                            | City        |
| 4        | Galactic Sector QQ7 Active J Gamma | Galaxy      |
| 5        |                                   | Planet      |
| 4        | Aldebaran                         | Star System |
+----------+-----------------------------------+-------------+
SELECT 4 rows in set (... sec)
```
```

```
cr> select sum(position) as sum_positions, kind from locations
... group by kind order by sum_positions;
+---------------+-------------+
| sum_positions | kind        |
+---------------+-------------+
| 10.0          | Star System |
| 13.0          | Galaxy      |
| 15.0          | Planet      |
| 29.0          | City        |
+---------------+-------------+
SELECT 4 rows in set (... sec)
```

分组(GROUP BY)

Crate支持group by子句。这个子句可以用来对结果行按一个或多个列的值进行分组。这意味着包含多个值的行将会合并在一起。 如果和聚合函数一起使用，这是很有用的:

```
cr> select count(*), kind from locations
... group by kind order by count(*) desc, kind asc;
+----------+-------------+
| count(*) | kind        |
+----------+-------------+
| 5        | Planet      |
| 4        | Galaxy      |
| 4        | Star System |
| 2        | City        |
+----------+-------------+
SELECT 4 rows in set (... sec)
```

> 注意 用作结果列或者order by子句的所有列必须在group by子句中使用。否则语句将不能执行。
>
> 注意 多值字段分组不起作用。如果在group by操作中有这样的字段将会抛出一个错误。
>
> 注意 分组可仅仅用于一个普通索引，默认对所有行。为了获取更多的信息，请参见Plain index(Default)。

HAVING

having子句等价于group by子句结果行的where子句。一个简单的having子句的例子使用一个等价的操作:

```
cr> select count(*), kind from locations
... group by kind having count(*) = 4 order by kind;
+----------+-------------+
| count(*) | kind        |
+----------+-------------+
|        4 | Galaxy      |
|        4 | Star System |
+----------+-------------+
SELECT 2 rows in set (... sec)
```

having子句的条件可以引用到group子句的结果列。而且可以在having子句中使用聚合就像早结果列中:

```
cr> select count(*), kind from locations
... group by kind having min(name) = 'Berlin';
+----------+------+
| count(*) | kind |
+----------+------+
|        2 | City |
+----------+------+
SELECT 1 row in set (... sec)
```

```
cr> select count(*), kind from locations
... group by kind having count(*) = 4 and kind like 'Gal%';
+----------+--------+
| count(*) | kind   |
+----------+--------+
|        4 | Galaxy |
+----------+--------+
SELECT 1 row in set (... sec)
```

比较运算符 having子句支持和where子句相同的操作:

操作符	描述
<	小于
>	大于
<=	小于等于
>=	大于等于
=	等于
<>	不等于
!=	不等于 等价于 <>
like	匹配给定值的部分
~	正则表达式匹配
!~	否定正则表达式匹配

REFRESH

Crate is eventually consistent. Data written with a former statement is not guaranteed to be fetched with the next following select statement for the affected rows.

Crate是最终一致的。前一个语句写入的数据不保证可以被下一个select语句获取这些受影响的行。

如果需要,可以显式刷新一个或者多个表以确保获取表的最新状态。

```
cr> refresh table locations;
REFRESH OK, 1 row affected (... sec)
```

一个以一个指定的间隔周期性地刷新。默认配置下，刷新间隔设置的是1000毫秒。一个表的刷新间隔可以通过表参数refresh_interval更改(参见refresh_interval)。

多表刷新(Multiple Table Refresh)

如果需要，可以在一个SQL语句中使用逗号分隔的方式定义多个表。这样确保它们都可以得到刷新，因此它们的数据集是一致的。如果每个给定表上的请求都完成，将会打印结果信息。

```
cr> REFRESH TABLE locations, parted_table;
REFRESH OK, 2 rows affected (... sec)
```

> 注意
> 如果一个或多个表或者分区不存在，则不会刷新任何给定的表/分区，而且将会返回一个错误。错误将返回第一个不存在的表/分区。

分区刷新(Partition Refresh)

此外，可以定义刷新一个分区表的分区(查看分区表)。

通过在refresh语句中使用PARTITION子句，可以分别执行给定分区的请求。这意味着只有指定的分区表的分区才可以被刷新。怎么在一个分区表上创建一个refresh请求的更多详情请参见SQL语法和概要(参见 刷新)。

```
cr> REFRESH TABLE parted_table PARTITION (day='2014-04-08');
REFRESH OK, 1 row affected (... sec)
```

如果省略PARTITION子句，则将刷新所有分区。如果一个表有多个分区，由于性能问题应该避免这么操作。

全文搜索(FULLTEXT SEARCH)

为了在一个或多个列上使用全文搜索，在创建列的时候必须定义带有分析器的全文索引：或者使用CREATE TABLE或ALTER TABLE ADD COLUMN。获取更多的信息请查看索引和全文索引。

MATCH谓词(MATCH Predicate)

概要(Synopsis)

```
MATCH (
     {  column_ident | ( column_ident [boost]  [, ...] ) }
 , query_term
)  [ using match_type [ with ( match_parameter [= value] [, ... ] ) ] ]
```

MATCH谓词在一个或多个索引列上执行全文搜索，而且支持不同的匹配技术。而且可以用来在geo_shape索引上执行地理搜索。

MATCH谓词的实际适用性取决于索引的类型。实际上，match_types和match_parameters的可用性依赖于具体索引。然而这一章，仅仅覆盖了在字符串列上MATCH谓词的使用。若要在geo_shape索引上使用MATCH，查看Geo Search。

若要在一个列上使用全文搜索，必须为这个列创建一个带分析器的全文索引。查看Indices和Fulltext Search获取更多详情。不同类型的索引和全文搜索有不同的目标，然而不能在一个MATCH谓词中查询不同索引类型的多个索引列。

获取匹配行的相关性，可以选择一个系统列_score。它包括一个和其它行的相关性的数字评分:值越高，行相关性越高。

```
cr> select name, _score from locations where match(name_description_ft, 'time') order
by _score desc;
+-----------+------------+
| name      |     _score |
+-----------+------------+
| Bartledan | 0.52200186 |
| Altair    | 0.4790727  |
+-----------+------------+
SELECT 2 rows in set (... sec)
```

MATCH以最简单的模式在一个单列上执行一个全文搜索。他使用query_term，若不提供分析器，则使用column_indet上设置分析器分析分词。分词器生成的token将和column_ident 索引匹配，若有一个匹配则MATCH将返回TRUE。

MATCH谓词也可以在带有一个单个query_term的多个列上执行全文搜索，而且可以给指定列添加权重，可以在每个column_ident上添加一个boost参数。与具有较高boost结果的列匹配，可以使此文档的_score值更高。

MATCH谓词也可以在多

match_type参数决定如何应用单个query_term以及如何计算结果_score。更多详情参见 Match Types。

结果默认按_score排序，但是可以添加ORDER BY子句覆盖。

参数(Arguments)

参数	描述
column_ident:	对索引列或者类型为字符串类型的已有列的引用。默认情况下并不仅仅存储原始数据，每个列都将被索引化，因此不使用全文索引在一个字符串类型的列上匹配时等价于使用“=”操作符。为了执行全搜索使用带分析器的索引。
boost:	一个列标识可以附带一个boost。是一个增加一个列和其他列之前平衡的一个权重因子。默认boost值为1。
query_term:	字符串将被分析而且分词结果将和索引比较。用于搜索的tokens使用布尔操作符OR组合，除非使用其他可选操作符
match_type:	可选的。默认为全文索引的best_fields。有关详情，请参阅匹配类型

> 注意
> MATCH谓词只能在WHERE子句而且使用户创建的表上使用。系统表将不支持使用MATCH谓词。
>
> 注意
> 一个MATCH谓词不能组合有连接关系的列。
>
> 注意
> 如果MATCH谓词不能用于有连接的两个关系的列，如果它们不能在逻辑上分别应用于它们中的每一个。例如:

```
This is allowed: FROM t1, t2 WHERE match(t1.txt, 'foo') AND match(t2.txt, 'bar');
But this is not: FROM t1, t2 WHERE match(t1.txt, 'foo') OR match(t2.txt, 'bar');
```

匹配类型(Match Types)

匹配类型query_term的应用和_score的创建，从而影响到被认为跟相关的那些文档。默认全文索引的match_type是best_fields。

	描述
best_fields:	使用最匹配的列的_score。例如如果一个列包含查询分词query_term中所有token，则将认为比仅包含一个的其他列更相关。如果省略，则此类型是默认的。
most_fields:	使用每一个匹配列的_score并使用它们的平均值
cross_fields:	这个匹配类型将query_term分析成token，并且立即在所有给定列中搜索，如果它们是一个大的列(假设它们具有相同的分析器)。所有的token必须至少出现在一列，因此查询"foo bar"应该在一列中具有标记"foo"而且"bar"在同一列或者其他列。
phrase:	此匹配类型和best_fields不同的是使用query_term构造查询短语。仅当列中的token与从query_term的中分析的列的顺序相同时，短语查询才会匹配。因此，查询"foo bar"(分析tokens:foo和bar)将仅仅匹配到包含这个顺序的两个token-之间没有其他token
phrase_prefix:	此类型和phrase类型大致一样，但是它允许在query_term的最后一个token上使用前缀匹配。例如，你查询"foo bar",则其中一列必须按顺序包含foo而且有一个token以ba开始。因此，一个包含"foo baz"的列将匹配，而且"foo bar"也一样。

选项(Options)

匹配选项进一步使用确切的匹配类型区分匹配过程的方式。不是所有的选项使用于所有匹配类型。有关详细信息，查看以下选项。

	描述
analyzer:	分析器将query_term转化为token。
boost:	此数字与匹配调用的结果评分相乘。如果此匹配调用与where子句猴子那个的其他条件一起使用。一个1.0以上的值将增加对整个查询的_score的影响，一个1.0以下的值将降低此影响
cutoff_frequency:	token的频率是一个token在一个列中出现的频率。这个选项指定一个最小的token频率，不包括来自整个具有很高_score的token。仅包括它们的_score若果一个较小频率的token也被匹配。此项配置将用于抑制像这样的高频分词引起的一个匹配。
fuzziness:	可用于执行模糊全文搜索。在数字列上使用数字，在时间戳列上使用一个龙类型的毫秒数，在string列上使用一个数字指定允许Levenshtein编辑的最大距离。使用prefix_length，fuzzy_rewrite和max_expansions微调模糊匹配过程。
fuzzy_rewrite:	和rewrite相同但是仅仅作用于使用模糊的查询。

max_expansions:	当使用fuzziness 或者 phrase_prefix时，这个选项控制一个token将会扩展出多少个可能的token。fuzziness 控制原始token和一组扩展的token之间有多大的距离或不同。选项控制此组可获得的大小。
minimum_should_match:	来自query_term的token数，当使用or时使用。默认为1。
operator:	可以是or或者and。默认是or。用于组合query_term的token。如果使用and，必须匹配每个出自query_term的token。如果使用or，仅仅匹配minimum_should_match数量必须匹配。
prefix_length:	当使用fuzziness选项 或者与带有phrase_prefix的选项时，这个选项控制被认为相似(同样的前缀或者模糊匹配距离/不同)的token的公共前缀的长度。
rewrite:	当使用phrase*prefix* ，前缀查询将使用所有可能分词来构造，并且将其写入到另一种查询中以计算其分数。可能的值有*constant_score_auto, constant_score_boolean, constant_score_filter, scoring_boolean,* `top_terms_N` , *top_terms_boost_N.* 常量*constant*...值可以和boost选项一起使用，以便为前缀匹配或模糊匹配的行设置一个常量_score。
slop:	当为短语匹配时，这个选项控制短语匹配的精确度(临近搜索)。如果设置为0(默认值),则分词必须按确切的顺序。如果两个转置的分词匹配，则应该设置一个最小的slop值2。仅仅应用phrase和phrase_prefix查询。就像一个slop 2的例子，查询“foo bar”将不仅匹配foo bar，还会匹配foo what bar。
tie_breaker:	当使用best_fields, phrase 或者 phrase_prefix，则其他列的_score将会乘以此值而且加到匹配最好的列的_score上。默认是0.0。不适用于most_fields类型，因为如果此类型被执行，就像是它的tie_breaker的值为1.0。
zero_terms_query:	如果query_term的分析结果没有生成token，则不会匹配任何文档。如果这里给的是all，则匹配所有文档。

用法(Usage)

使用MATCH谓词进行全文搜索:

```
cr> select name from locations where match(name_description_ft, 'time') order by _scor
e desc;
+-----------+
| name      |
+-----------+
| Bartledan |
| Altair    |
+-----------+
SELECT 2 rows in set (... sec)
```

和搜索字符串匹配到的行将返回TRUE。更多详情查看等值匹配，相关的列，_score可以被选中:

```
cr> select name, _score
... from locations where match(name_description_ft, 'time') order by _score desc;
+-----------+------------+
| name      |     _score |
+-----------+------------+
| Bartledan | 0.52200186 |
| Altair    | 0.4790727  |
+-----------+------------+
SELECT 2 rows in set (... sec)
```

> 注意
> _score不是一个绝对值。它只是相对于其它行的一个设置。

如果一个搜索跨多个列的内容则应该有两种可能:

在你的表中使用一个复杂的索引列。查看一个复杂索引的定义。
在多列上使用MATCH谓词
当查询多列时，有许多方式计算相关度 _score。这些不同的技术讲座匹配类型。

为了增加一个列匹配非常好的行相关度，使用best_fields(默认)。如果匹配比较好的行遍布所有包含的列，则应该有更多的相关度，使用most_fields。要想像使用一列一样查询多行，使用cross_fields。为了查询匹配的短语使用phrase或者phrase_prefix:

```
cr> select name, _score from locations
... where match(
...     (name_description_ft, race['name'] 1.5, kind 0.75),
...     'end of the galaxy'
... ) order by _score desc;
+------------------+------------+
| name             |     _score |
+------------------+------------+
| NULL             | 0.91878563 |
| Altair           | 0.11727207 |
| Aldebaran        | 0.10617954 |
| North West Ripple | 0.06788038 |
| Outer Eastern Rim | 0.06788038 |
+------------------+------------+
SELECT 5 rows in set (... sec)
```

```
cr> select name, description, _score from locations
... where match(
...     (name_description_ft), 'end of the galaxy'
... ) using phrase with (analyzer='english', slop=4);
+------+-------------------------+----------+
| name | description             |   _score |
+------+-------------------------+----------+
| NULL | The end of the Galaxy.% | 2.296235 |
+------+-------------------------+----------+
SELECT 1 row in set (... sec)
```

有大量的选项来优化你现有的全文搜索。一个详细的引用可以在查看MATCH谓词。

否定搜索(Negative Search)

A negative fulltext search can be done using a NOT clause:
否定搜索可以使用NOT子句来实现:

```
cr> select name, _score from locations
... where not match(name_description_ft, 'time')
... order by _score, name asc;
+-----------------------------------+--------+
| name                              | _score |
+-----------------------------------+--------+
|                                   |    1.0 |
| Aldebaran                         |    1.0 |
| Algol                             |    1.0 |
| Allosimanius Syneca               |    1.0 |
| Alpha Centauri                    |    1.0 |
| Argabuthon                        |    1.0 |
| Arkintoofle Minor                 |    1.0 |
| Galactic Sector QQ7 Active J Gamma |    1.0 |
| North West Ripple                 |    1.0 |
| Outer Eastern Rim                 |    1.0 |
| NULL                              |    1.0 |
+-----------------------------------+--------+
SELECT 11 rows in set (... sec)
```

通过_score过滤(Filter By _score)

可以通过_score列来过滤结果，但是它的值是一个相对于所有结果中最高分而计算的值，因此从来不是绝对的或者可比较的，在排序之外的使用性是非常有限的。在_score列上使用“>=”操作符过滤将不会有什么意义而且可能导致不可预测的结果集。

无论若何，我们在这里作一个示范:

```
cr> select name, _score
... from locations where match(name_description_ft, 'time')
... and _score >= 0.8 order by _score;
+-----------+------------+
| name      |     _score |
+-----------+------------+
| Bartledan | 0.8679331  |
| Altair    | 0.88735807 |
+-----------+------------+
SELECT 2 rows in set (... sec)
```

你可能注意到这点，_score值对同样查询文本和文档的值已经改变，因为它是一个相对所有结果的比率，通过_score过滤，‘all results’已经改变。

> 警告
> 如上所述，_score是一个相对数字而且在搜索中不可比。因此，极大阻止了过滤。

地理信息搜索(GEO SEARCH)

可以使用Crate的geo_point和geo_shape数据类型来存储和查询很多种类的地理信息。利用这些支持可以用来存储地理位置，道路，形状，区域和其他实体信息。这些信息可以用来查询距离，容量，马路口等，使有可能创建具有丰富地理特性的app和服务。

地理形状使用指定索引存储。地理位置点用它们的坐标来表示。它们表示为相应数据类型的列。

在geo_shape列上使用地理位置索引，以便加速对地理位置甚至复杂形状的搜索。这个索引过程会导致不准确的呈现(查看geo_shape获取更多详情)。Crate不能操作形状矢量，仅仅是一种具有给定精度的格子。

创建包含地理信息的表很简单:

```
cr> CREATE TABLE country (
...   name string,
...   country_code string primary key,
...   shape geo_shape INDEX USING 'geohash' WITH (precision='100m'),
...   capital string,
...   capital_location geo_point
... ) WITH (number_of_replicas=0);
CREATE OK (... sec)
```

这个表将包含一个国家的形状和它的首都的位置以及其他元数据。使用最大精度为100米的geohash索引来索引化位置信息(更多信息，查看Geo形状索引数据结构)。

插入Austria:

```
cr> INSERT INTO country (mame, country_code, shape, capital, capital_location)
... VALUES (
...  'Austria',
...  'at',
...  {type='Polygon', coordinates=[[
...        [16.979667, 48.123497], [16.903754, 47.714866],
...        [16.340584, 47.712902], [16.534268, 47.496171],
...        [16.202298, 46.852386], [16.011664, 46.683611],
...        [15.137092, 46.658703], [14.632472, 46.431817],
...        [13.806475, 46.509306], [12.376485, 46.767559],
...        [12.153088, 47.115393], [11.164828, 46.941579],
...        [11.048556, 46.751359], [10.442701, 46.893546],
...        [9.932448,  46.920728], [9.47997,   47.10281 ],
...        [9.632932,  47.347601], [9.594226,  47.525058],
...        [9.896068,  47.580197], [10.402084, 47.302488],
...        [10.544504, 47.566399], [11.426414, 47.523766],
...        [12.141357, 47.703083], [12.62076,  47.672388],
...        [12.932627, 47.467646], [13.025851, 47.637584],
...        [12.884103, 48.289146], [13.243357, 48.416115],
...        [13.595946, 48.877172], [14.338898, 48.555305],
...        [14.901447, 48.964402], [15.253416, 49.039074],
...        [16.029647, 48.733899], [16.499283, 48.785808],
...        [16.960288, 48.596982], [16.879983, 48.470013],
...        [16.979667, 48.123497]]
...  },
...  'Vienna',
...  [16.372778, 48.209206]
... );
INSERT OK, 1 row affected (... sec)
```

> 注意
>
> 地理形状必须使用ISO 19107来完全校验。如果你有导入geo数据的问题，他们可能没有完全校验。在大多数情况下，他们将被使用下面的工具来修复:https://github.com/tudelft3d/prepair

地理位置点可以作为一个带有lon和lat的double数组，如上所示或者作为WKT的字符串。

地理形状可以被当做GeoJSON对象或者参数插入，如上所示和WKT的字符串。

当涉及到对你的地理数据有一些有意义的洞察是，Crate支持不同类型的地理查询。

快速查询和利用地理索引的方式是使用Match谓词:

Match Predicate

MATCH谓词可以用来在索引或者索引列上执行多种查询。同时他也可用来执行全文搜索在string类型的分析索引上，有人可以在地理索引上操作，查询地理新装和点的关系。

MATCH (column_ident, query_term) [using match_type]

地理搜索的匹配谓词支持将geo_shape索引列的单个column_ident作为第一个参数。

第二个参数,query_term被用来在索引geo_shape之间匹配。

匹配操作有匹配类型match_type决定，match_type将决定我们想要匹配的空间关系。可用的match_type有:

match_type	详情
(相交)intersects:	(默认)如果两个形状共享一些点和或者区域,它们是相交的。并且认为会用此匹配类型。这也排除了遏制或完全平等
(不相交)disjoint:	如果两个形状没有共享的点或者区域，则他们是不相交的。这是相交的反面。
(包含)within:	如果索引列column_ident 的形状完全在query_term形状里面,则认为它们使用此匹配类型

> 注意
> 匹配谓词只能在where子句中使用而且必须为用户创建的表。在MATCH谓词不支持对系统表的操作。
>
> 注意 一个谓词不能组合所有一个join关系的所有列。
>
> 注意 MATCH谓词不能用于连接的两个关系的列，如果它们不能在逻辑上分别应用于它们中的每一个。例如: 这个是允许的: FROM t1, t2 WHERE match(t1.shape, 'POINT(1.1 2.2)') AND match(t2.shape, 'POINT(3.3 4.4)'); 但是这个不允许: FROM t1, t2 WHERE match(t1.shape, 'POINT(1.1 2.2)') OR match(t2.shape, 'POINT(3.3 4.4)');

```
cr> SELECT name from countries
... WHERE
...  match(
...   shape,
...   'LINESTRING (13.3813 52.5229, 11.1840 51.5497, 8.6132 50.0782, 8.3715 47.94
57, 8.5034 47.3685)'
... );
+------+
| name |
+------+
+------+
SELECT 0 rows in set (... sec)
```

```
cr> SELECT name from countries
... WHERE
...  match(
...   shape,
...   'LINESTRING (13.3813 52.5229, 11.1840 51.5497, 8.6132 50.0782, 8.3715 47.94
57, 8.5034 47.3685)'
...  ) USING disjoint;
+---------+
| name    |
+---------+
| Austria |
+---------+
SELECT 1 row in set (... sec)
```

精确查询(Exact Queries)

精确查询使用下面标准函数:

- intersects(geo_shape, geo_shape) returns boolean
- within(shape1, shape2) returns boolean
- distance(geo_point1, geo_point2) returns double

他们是精确的，但是这会带来一些性能代价。他们不使用索引但能在GeoJSON上工作，插入计算的形状向量。这个访问代价相当高而且可能减您的查询。为了查询更快，使用Match谓词。

但在有限结果集上执行，他们将帮助您精确洞察您的地理数据:

```
cr> SELECT within(capital_location, shape) AS capital_in_country
... FROM countries ORDER BY country_code;
+--------------------+
| capital_in_country |
+--------------------+
| TRUE               |
+--------------------+
SELECT 1 row in set (... sec)
cr> SELECT distance(capital_location, [0, 0]) as from_northpole
... FROM countries ORDER BY country_code;
+--------------------+
| from_northpole     |
+--------------------+
| 1234.2345          |
+--------------------+
SELECT 1 row in set (... sec)
```

```
cr> SELECT intersects(
...   {type='LineString', coordinates=[[13.3813, 52.5229],
                                       [11.1840, 51.5497],
                                       [8.6132,  50.0782],
                                       [8.3715,  47.9457],
                                       [8.5034,  47.3685]]},
...   shape) as berlin_zurich_intersects
... FROM countries ORDER BY country_code;
+--------------------------+
| berlin_zurich_intersects |
+--------------------------+
| FALSE                    |
+--------------------------+
SELECT 1 row in set (... sec)
```

尽管如此，这些标准函数可以在允许使用标准函数的SQL查询中的任何地方使用。