

**KBase 非结构化数据库管理系统**

**K-SQL 10参考大全**



**同方知网（北京）技术有限公司**

**2014年11月**

**版权**

本手册著作权为同方知网（北京）技术有限公司，©2005版权所有，保留一切权力。

未经同方知网（北京）技术有限公司的书面许可，不得为任何目的、以任何形式或手段复制或传播本手册的任何部分。

**商标**

CNKI、“中国知识基础设施工程”、TTNK、“同方光盘”、“中国知网”及其标志均为同方知网（北京）技术有限公司的商标。

Adobe和Acrobat是AdobeSystemsIncorporated在美国等其它国家（或地区）的注册商标或商标。

Microsoft Windows®、Windows® 2000、Windows® Millennium、Windows®  NT、Windows® XP及Windows® Server 2003 均为微软公司的注册商标。

Intel、Pentium 及 MMX 均为英特尔公司的注册商标。

所有其它公司名称或产品名称均为其各自所有者的商标或注册商标或服务商标，藉以识别该所有者。

所有在本手册使用的商标为该商标所有人的资产

**限制条件及免责申明**

鉴于软件技术发展迅速，产品不断升级，虽然本公司对说明的内容将尽量及时随之修改，但是也存在可能来不及修改的情况。因此本公司保留在必要时对本手册的内容进行修改的权利，如果本手册的内容发生变动，恕不另行通知。

本手册例子中所用的公司、人名或数据若非特别指明，均属虚构。

同方知网（北京）技术有限公司

**软件最终用户许可协议**

**（TTNK SEULA）**

**重要须知**

请认真阅读：本最终用户许可协议（《协议》）是您（个人或单一实体）与同方知网（北京）技术有限公司（以下简称**TTNK**）之间有关本《协议》随附的 TTNK 软件（包括相关媒体和 TTNK 基于 Internet 的服务，统称为“软件”）的法律协议。本《协议》的一份修正条款或补充条款可能随“软件”一起提供。您一旦安装、复制或使用“软件”，即表示您同意接受本《协议》各项条款的约束。如果您不同意本《协议》中的条款，请不要安装、复制或使用“软件”；您可在适用的情况下将其退回原购买处，并获得全额退款。

1. **许可证的授予**

TTNK 授予您以下权利，条件是您遵守本《协议》的各项条款和条件：

1.1 **安装和使用**

您只能在一台个人计算机或其他设备上安装和使用“软件”的一个副本；

1.2 **媒体元素的许可授予**

“软件”中可能会包括一些标明供您使用的照片、剪贴画、图形、动画、声音、音乐和视频剪辑（统称为“媒体元素”）。您可以复制和修改“媒体元素”，并将其与您所作的修改一起作为您的软件产品和服务（包括您的 Web 站点）的一部分进行许可、显示和分发，但不得进行以下任何一种活动：

* 如果产品或服务的主要价值在于“媒体元素”，则不得将这些“媒体元素”的副本本身或者将其作为任何集合、产品或服务的组成部分进行销售、许可或分发。
* 不得给您产品或服务的客户授予任何许可或分发“媒体元素”的权利。
* 如果“媒体元素”中包含可识别的个人、政府、徽标、缩写、徽章、商标或实体的表现形式，不得许可或分发该“媒体元素”以用于任何商业目的，亦不得明示或默许任何认可或与任何产品、服务、实体或活动的关联。
* 不得使用“媒体元素”创建淫秽或诽谤性的作品（依据创建作品时适用法律中的规定细则）。

此外，您必须：

(a) 赔偿 TTNK 并使其免受由于许可、使用或分发经您修改的“媒体元素”所发生或引起的任何索赔或诉讼（包括律师费），并且

(b) 在您的包含“媒体元素”的产品和服务上附加一个有效的著作权声明。

1.3 **文档的许可授予**

“软件”随附的文档仅为内部、非商业性参考目的授予许可。

* 1. **模板的许可授予**

“软件”中可能包含网站模板。您可以对本《协议》随附的 TTNK 软件所提供的网站模板进行复制和修改，并将这些模板与您的修改一起分发，以供“软件”的其他被许可人使用。您还可以对通过基于 Internet 的相关服务所获得的模板进行复制和修改，并与您的修改一同分发，以供“软件”的其他被许可人使用，但仅限于涉及私人间通信的个人或商业联络。您不被许可进行以下任何活动之一：

* 不得销售、再销售、许可、出租、租赁、出借或以其他方式有偿转让模板。
* 不得将通过基于 Internet 的服务所获得的模板作为任何产品或服务的组成部分进行分发。
* 不得将任何通过基于 Internet 的服务所获得的模板在任何网络计算机上复制或公开，或通过任何媒体方式传播。

您必须赔偿 TTNK 并使其免受由于许可或分发经您修改的模板所发生或引起的任何索赔或诉讼（包括律师费）。

2. **基于 Internet 的服务**

对使用任何与“软件”相关的 TTNK 的基于 Internet 的服务，如果其使用方式有可能损坏、禁用、削弱这些服务或加重其负担，或干扰任何第三方使用和享受这些服务，则您不能使用这些服务。您不得未经授权试图访问与基于 Internet 的服务相关的任何服务、帐户、计算机系统或网络。

3. **权利的保留和所有权**

TTNK 保留未在本《协议》中明示授予您的一切权利。“软件”受著作权和其他知识产权法律及条约的保护。TTNK 或其供应商拥有“软件”的产权、著作权和其他知识产权。“软件”只授予使用许可，而非出售。本《协议》不授予您任何使用 TTNK 商标或服务标记的权利。

4. **对反向工程、反编译和反汇编的限制**

您不得对“软件”进行反向工程、反编译或反汇编；尽管有此项限制，但如果适用法律明示允许上述活动，并仅在适用法律明示允许上述活动的范围内，则不在此限。

5. **无出租/商业宿主服务**

您不得出租、租赁、出借“软件”或以“软件”提供商业宿主服务。

6. **至第三方站点的链接**

TTNK 不对任何第三方站点或服务的内容、第三方站点或服务包含的任何链接或第三方站点或服务的任何更改或更新负责。TTNK 提供这些到第三方站点和服务的链接和访问只是为了您的方便，对包含的任何链接和访问并不意味着 TTNK 对第三方站点或服务的认可。

7. **附加软件/服务**

除非我们随“软件”的更新、补充、附加组件或基于 Internet 的服务组件一起提供单独的条款，否则本《协议》适用于 TTNK 在您获得“软件”的初始副本之日后可能提供给您的或为您准备的“软件”的更新、补充、附加组件或基于 Internet 的服务组件。就通过使用“软件”而提供给您的或为您准备的任何基于 Internet 的服务而言，TTNK 保留停止这类服务的权利。

8. **升级版本**

要使用标明为升级版本的“软件”，您必须首先获得由 TTNK 标明为适合使用升级版本的软件的使用许可。安装升级版本之后，不得继续使用构成适合升级基础的原始软件，除非该原始软件作为升级后软件的一部分。

9. **非再销售性软件**

标明为“非再销售品”的“软件”不得销售或以其他方式有偿转让，不得用于演示、测试或评估之外的任何其他目的。

10. **学术版软件**

要使用标明为“学术版”的“软件”，您必须是一个“合格的教育用户”。如果您有与资格相关的问题，请与 TTNK 销售信息中心联系。

11. **出口限制**

您承认“软件”受中国出口法律管辖。您同意遵守所有适用于“软件”的适用的国际法和国内法，其中包括《中国出口管理条例》，以及由中国和其他国家（地区）政府颁发的最终用户、最终使用和目的地方面的限制。

12. **软件转让**

内部。您可以将“软件”副本转移到另一台设备。转移之后，必须将“软件”从原设备上完全删除。转让给第三方。如果您是“软件”的原始许可用户，您可以将本《协议》、“软件”和真品证书（如果适用）一次性永久转让给另一个最终用户，条件是您不保留“软件”的任何副本。此转让必须包括“软件”的所有部分（包括全部组件、媒体和印刷资料、任何升级版本、本《协议》以及（如果适用）真品证书）。这种转让不得为非直接转让，如以寄售方式转让。在转让之前，接收“软件”的最终用户必须同意本《协议》的各项条款。

13. **终止**

如果您未遵守本《协议》的各项条款和条件，在不损害任何其他权利的情况下，TTNK 可终止本《协议》。如此类情况发生，您必须销毁“软件”的所有副本及其全部组成部分。

1. **全部协议可分开执行**

这份《协议》（包括随“软件”提供的本《协议》的任何补充条款或修正条款）是您与 TTNK 之间就“软件”和支持服务（如果有）达成的全部协议，并且取代“软件”或本《协议》中所包含的任何主题事项的所有先前的或同时存在的口头或书面的通信、建议和声明。如果任何 TTNK 的支持服务的政策或计划的条款与本《协议》的条款有冲突，以本《协议》中的条款为准。如果本《协议》的任何条款被认定为作废、无效、不能执行或非法，其他条款继续完全有效。

如你对本最终用户许可协议有任何问题，或如你因任何原因希望联络 TTNK，请使用随本产品附上的地址联络 TTNK 北京总部或各地分公司（办事处），或在互联网上访问 TTNK，网址为：[http://www.cnki.net](http://www.cnki.net/)

**目 录**

[第一章 KSQL简介 1](#_Toc403566233)

[1、字符集 1](#_Toc403566234)

[2、KSQL保留字与标识符 1](#_Toc403566235)

[3、KSQL语句的功能分类 2](#_Toc403566236)

[第二章 KSQL运算符 3](#_Toc403566237)

[1、逻辑运算符 3](#_Toc403566238)

[2、复合运算符 3](#_Toc403566239)

[3、比较运算符 3](#_Toc403566240)

[4、检索通配符 3](#_Toc403566241)

[5、位置描述符 3](#_Toc403566242)

[6、数据分隔符 4](#_Toc403566243)

[7、匹配运算符 4](#_Toc403566244)

[8、模糊匹配运算符 4](#_Toc403566245)

[9、相关匹配运算符 4](#_Toc403566246)

[第三章 KSQL数据类型 6](#_Toc403566247)

[1、数据存储类型概述 6](#_Toc403566248)

[2、数据存储类型特性 7](#_Toc403566249)

[2.1 INTEGER、NUM和INT64类型 7](#_Toc403566250)

[2.2 AUTO类型 7](#_Toc403566251)

[2.3 DATE 和 TIME类型 7](#_Toc403566252)

[2.3 CHAR 和 VCHAR类型 8](#_Toc403566253)

[2.4 MVCHAR类型 8](#_Toc403566254)

[2.5 LMVCHAR类型 9](#_Toc403566255)

[2.6 MTEXT 和 LTEXT类型 9](#_Toc403566256)

[2.7 SOB、LOB和DOB类型 9](#_Toc403566257)

[2.8 VECTOR类型 9](#_Toc403566258)

[2.9 DOCUMENT / DOCPATH类型 9](#_Toc403566259)

[2.10 VIRTUAL类型 9](#_Toc403566260)

[2.11 SNAPSHOT等类型 10](#_Toc403566261)

[3、数据索引类型概述 10](#_Toc403566262)

[4、数据索引类型特性 11](#_Toc403566263)

[4.1 INTEGER、QINTEGER 类型 11](#_Toc403566264)

[4.2 INT64类型 12](#_Toc403566265)

[4.3 NUM类型 12](#_Toc403566266)

[4.4 DATE、TIME和QDATE类型 13](#_Toc403566267)

[4.5 CHAR和 ECHAR类型 14](#_Toc403566268)

[4.6 MVCHAR和 EMVCHAR类型 17](#_Toc403566269)

[4.7 WORD类型 17](#_Toc403566270)

[4.8 STRING和 STRCHAR类型 18](#_Toc403566271)

[4.9 MSTRCHAR类型 20](#_Toc403566272)

[4.10 TEXT、TEXTCHAR、LTEXT和LTEXTCHAR类型 21](#_Toc403566273)

[4.11 TITLE、EXTITLE、EXTITLECHAR、QSTRING、QSTRCHAR类型 24](#_Toc403566274)

[4.12 ABSTRACT类型 25](#_Toc403566275)

[4.13 COMPACTQTEXT类型 26](#_Toc403566276)

[4.14 QTEXT类型 27](#_Toc403566277)

[4.15 LFTEXT类型 28](#_Toc403566278)

[4.16 TYTDTEXT和TYTEXT类型 28](#_Toc403566279)

[4.17 VECTOR类型 29](#_Toc403566280)

[4.18 FINGERPRINT类型 29](#_Toc403566281)

[4.19 SEARCHENGINE类型 29](#_Toc403566282)

[4.20 MIX类型 29](#_Toc403566283)

[5、数据索引模式 29](#_Toc403566284)

[5.1 NON（无索引） 30](#_Toc403566285)

[5.2 NORMAL（普通索引） 30](#_Toc403566286)

[5.3 UNIQ（单一索引） 30](#_Toc403566287)

[5.4 STEM（词干索引） 30](#_Toc403566288)

[第四章 KSQL数据定义语句 31](#_Toc403566289)

[1、数据库定义语句 31](#_Toc403566290)

[1.1创建数据库 31](#_Toc403566291)

[1.2删除数据库 31](#_Toc403566292)

[2、表定义语句 31](#_Toc403566293)

[2.1创建表 31](#_Toc403566294)

[2.2修改表结构 33](#_Toc403566295)

[2.3修改表的物理路径 34](#_Toc403566296)

[2.4删除表 35](#_Toc403566297)

[2.5表的高级操作 35](#_Toc403566298)

[2.6创建联合表 38](#_Toc403566299)

[3、视图定义语句 39](#_Toc403566300)

[3.1创建视图 39](#_Toc403566301)

[3.2删除视图 39](#_Toc403566302)

[3.3视图嵌套 39](#_Toc403566303)

[4、索引定义语句 40](#_Toc403566304)

[4.1建立索引 40](#_Toc403566305)

[4.2删除索引 40](#_Toc403566306)

[4.3激活/禁止表的索引 41](#_Toc403566307)

[5、存储空间定义语句 41](#_Toc403566308)

[5.1 定义存储空间 41](#_Toc403566309)

[5.2 删除存储空间 43](#_Toc403566310)

[6、索引空间定义语句 43](#_Toc403566311)

[6.1 创建一个索引空间 43](#_Toc403566312)

[6.2 修改一个或多个索引的索引空间位置 43](#_Toc403566313)

[7、字段映射关系定义语句 44](#_Toc403566314)

[7.1 添加字段映射关系 44](#_Toc403566315)

[7.2删除字段映射关系 44](#_Toc403566316)

[第五章 KSQL数据查询语句 45](#_Toc403566317)

[1、单表查询 45](#_Toc403566318)

[2、多表查询 47](#_Toc403566319)

[3、跨库查询 47](#_Toc403566320)

[4、排序查询 50](#_Toc403566321)

[5、分组查询 51](#_Toc403566322)

[6、过滤查询 51](#_Toc403566323)

[7、嵌套过滤查询 52](#_Toc403566324)

[8、其他 KSQL高级查询 53](#_Toc403566325)

[8.1聚集函数查询 53](#_Toc403566326)

[8.2复杂组合查询 54](#_Toc403566327)

[8.3自动扩展查询 54](#_Toc403566328)

[8.4模糊查询 55](#_Toc403566329)

[8.5 BELONG TO查询 55](#_Toc403566330)

[8.6 函数查询 56](#_Toc403566331)

[8.7 分页查询 57](#_Toc403566332)

[第六章 KSQL数据操作语句 59](#_Toc403566333)

[1、插入记录 59](#_Toc403566334)

[2、更新记录 59](#_Toc403566335)

[3、删除记录 60](#_Toc403566336)

[4、重整表 60](#_Toc403566337)

[5、清空表 60](#_Toc403566338)

[6、文档存储类型管理 61](#_Toc403566339)

[6.1 创建文本抽取器 61](#_Toc403566340)

[6.2 删除文本抽取器 61](#_Toc403566341)

[6.3 文档型数据插入格式 61](#_Toc403566342)

[第七章 KSQL统一扩展管理语句 62](#_Toc403566343)

[1、排序词典与排序索引管理 62](#_Toc403566344)

[1.1排序词典管理 62](#_Toc403566345)

[1.2表排序文件管理 63](#_Toc403566346)

[1.3表排序文件同步 65](#_Toc403566347)

[2、大内存文件缓存 65](#_Toc403566348)

[2.1手动配置 66](#_Toc403566349)

[2.2 使用热点统计分析工具自动配置 67](#_Toc403566350)

[2.3 AWE 参数说明 68](#_Toc403566351)

[2.4 其他参数说明 69](#_Toc403566352)

[3、检索缓存 69](#_Toc403566353)

[3.1参数值说明 69](#_Toc403566354)

[3.2参数值设置 69](#_Toc403566355)

[4、集群 70](#_Toc403566356)

[4.1 集群角色介绍 70](#_Toc403566357)

[4.2集群参数配置 70](#_Toc403566358)

[5、其他杂项 71](#_Toc403566359)

[5.1 系统变量设置 71](#_Toc403566360)

[5.2 不同分词类型的表实现兼容的办法 71](#_Toc403566361)

[5.3 排序文件同步时间设置 72](#_Toc403566362)

[5.4 表读写分离应用设置 72](#_Toc403566363)

[5.5 数据排重 73](#_Toc403566364)

[第八章 KSQL批量数据处理语句 74](#_Toc403566365)

[1、排重合并表 74](#_Toc403566366)

[2、批量导入数据 74](#_Toc403566367)

[3、批量导出数据 76](#_Toc403566368)

[4、批量更新方法 76](#_Toc403566369)

[5、脚本更新方法 78](#_Toc403566370)

[5.1 创建优化的主键索引 78](#_Toc403566371)

[5.2 释放优化的主键索引 79](#_Toc403566372)

[5.3 批量插入记录 79](#_Toc403566373)

[5.4 批量修改记录 80](#_Toc403566374)

[5.5 批量删除记录 80](#_Toc403566375)

[5.6 批量替换记录 80](#_Toc403566376)

[5.7 设置批量处理的数据文件 81](#_Toc403566377)

[5.8 脚本批量处理实例 81](#_Toc403566378)

[第九章 KSQL文本处理语句 83](#_Toc403566379)

[1、文本处理 KSQL概述 83](#_Toc403566380)

[1、文本挖掘引擎启动 84](#_Toc403566381)

[2、文本挖掘引擎结束 84](#_Toc403566382)

[3、创建知识域 84](#_Toc403566383)

[4、删除知识域 85](#_Toc403566384)

[5、启动知识域 85](#_Toc403566385)

[6、关闭知识域 86](#_Toc403566386)

[7、特征词分布规律分析 86](#_Toc403566387)

[8、文本分类器 87](#_Toc403566388)

[9、信息过滤器 88](#_Toc403566389)

[10、文本分类 89](#_Toc403566390)

[11、信息过滤 91](#_Toc403566391)

[12、相似索引 92](#_Toc403566392)

[13、文本聚类 94](#_Toc403566393)

[14、关键词提取器 95](#_Toc403566394)

[15、自动摘要提取器 97](#_Toc403566395)

[16、内置引擎关键词提取 98](#_Toc403566396)

[17、内置引擎文本分类 99](#_Toc403566397)

[附 录 101](#_Toc403566398)

[附录一： KSQL的分词规则 101](#_Toc403566399)

[1、切词器 101](#_Toc403566400)

[2、命名实体抽取器 101](#_Toc403566401)

[附录二： KSQL的保留字 101](#_Toc403566402)

[附录三： 汉字编码 ＧＢＫ 102](#_Toc403566403)

[附录四：样本数据库 103](#_Toc403566404)

第一章 KSQL简介

结构化查询语言SQL(Structured Query Language)是一种关系数据库语言，最早提出是在1974年。由于SQL语言接近英语的语句结构，易于理解、清晰简洁、方便灵活、功能强大，倍受用户及计算机工业界的欢迎，被众多计算机公司和数据库厂商所采用，如IBM，ORALCE，Microsoft等。经各公司的不断修改、扩充和完善，SQL语言最终发展成为关系数据库的标准语言，国内外各大型数据库系统均支持SQL语句。

SQL语言的影响不仅在数据库领域，在全文检索、CAD、软件工程、人工智能等领域，不仅把SQL作为检索数据的语言规范，而且也把SQL作为检索文本、图形、图象、声音、文字、知识等信息类型的语言规范。

KSQL语言是KBase数据库的查询语言，不仅包含了SQL语言的基本语句，而且根据文档检索（尤其是中文检索）特定需求，对其进行了扩充。因此，KSQL强大的检索功能能够满足诸多文档检索的需求，同时也支持基本的关系数据检索，是一种功能强大的检索语言。

KSQL是一种大小写不敏感的数据检索语言，通过自动分词，以及数据类型、索引上的支持，提供了强大的全文检索功能，尤其是其中文的检索能力。

KSQL还支持多种查询方式，包括精确查找、包含查找、前方一致查询、位置查询、比较查询、同义词查询、复合查询等，从而使得用户可以写出非常复杂的查询语句。

为了支持复杂的查询，KSQL允许在一个字段内通过分隔符来存放多个值，这虽然与传统的数据库第一范式相违背，但对于查询能力来说，却是一个创造性的提高。

总之，KSQL的侧重点在于丰富和完善查询的功能，因此其数据类型和查询、定义语句都是围绕这一中心主题来设计的，强大的中文检索能力是其一大特色，在学习和理解KSQL时，要始终明确这一主题。

1、字符集

KBase 数据库系统中的字符可以是数字、字母、专用字符和汉字，在系统中用于定义语言的终结符号与字符串元素。

数字是指从0到9的阿拉伯数字。

字母是指英文字母，包括大写字母(如 A，Z)和小写字母(如a，z)。

专用字符可以是实现者定义的除数字和字母字符外的任意字符。专用字符应包括在SQL语言最终产生式中出现的除数字和字母以外的所有符号，另外还包括百分比号与下划线字符。

KBASE 系统处理的字符类型：一般字段类型完全支持GBK编码、UTF-8编码。

2、KSQL保留字与标识符

KSQL中标识符分为正规标识符和定界标识符两大类。

正规标识符以字母、下划线或汉字开头，后面可以跟随字母、数字、下划线或者汉字，正规标识符的最大长度是256个英文字符或128个汉字。正规标识符不能是保留字。

正规标识符的例子：A，test1，\_TABLE\_B，表1。

定界标识符的标识符体用英文单引号和双引号括起来时，标识符体可以包含任意字符，也可以包含保留字。

定界标识符的例子："table"，"A"，"!@#$"。

如果定界标识符的标识符体中包含英文单引号和双引号，则需要特别方式处理。处理方法一是，仅含有单引号的标识符可以用双引号引起，仅含有双引号的标识符可以用单引号引起；方法二是采用转义的方式，转义表如下：

|  |  |
| --- | --- |
| **符号** | **转义形式** |
| \ | \\ |
| ’ | \’ |
| ” | \” |

保留字的清单参见附录三。

|  |
| --- |
| **注意：**如果标识符包含了标识符分隔符，如前面所述的英文单引号、双引号，或者是空格、KSQL的操作符等内容，则必须将其用引号引起来。 |

3、KSQL语句的功能分类

根据功能的不同，KSQL的语句包括数据定义语句、数据查询语句、数据操纵语句，各部分的内容为：

1. 数据定义语句

包括数据库的创建和删除、表的创建、修改和删除、视图的创建和删除、索引的创建和删除。

2. 数据查询语句

包括各种查询语句，如精确查询、模糊查询、多表查询、连接查询、分组查询、排序查询等。

3. 数据操纵语句

包括对插入记录、删除记录、更新记录、重整表、计算全字段的外部因子等。

4. 统一扩展管理语句

包括所有以关键字 DBUM 开头的 对SQL做了大量扩展的其他KBase语句。

第二章 KSQL运算符

本章对KSQL中的运算符做一下集中的介绍，KSQL中的运算符有逻辑运算符、比较运算符、复合列值运算符、连接操作符、检索通配符、位置描述符、数据分隔符和模糊匹配符等。

## 1、逻辑运算符

|  |  |
| --- | --- |
| **符号** | **意义** |
| AND | 逻辑“与” |
| OR | 逻辑“或” |
| NOT | 逻辑“非” |

逻辑运算符主要用于KSQL语句的WHERE子句，用于查询条件的各种逻辑组合。

## 2、复合运算符

|  |  |
| --- | --- |
| **符号** | **意义** |
| + | 逻辑“或” |
| - | 逻辑“非” |
| \* | 逻辑“与” |

复合运算符主要用于检索关键字的复合表示，可以表达复杂、高效的检索语句，并且可以简化KSQL语句（相对于使用逻辑运算符）。

## 3、比较运算符

|  |  |
| --- | --- |
| **符号** | **意义** |
| > | 大于 |
| < | 小于 |
| >= | 大于等于 |
| <= | 小于等于 |

在KSQL中没有不等于比较运算符，有不等于的要求时，可以采用逻辑操作符“NOT”或者复合列值运算符“-”通过构造逻辑表达式来代替。

## 4、检索通配符

|  |  |
| --- | --- |
| **符号** | **意义** |
| ? | 相隔1个字符 |
| \* | 相隔0个或多个字符 |

检索通配符主要用于CHAR、VCHAR、MVCHAR、WORD、STRING、STRCHAR等类型索引的检索匹配。在特定的索引类型中，这些检索通配符意义不尽相同，如在CHAR索引中 ?\\* 还用于包含查找和前方一致查找，这时 ?和\*含义相同，这种处理主要是为了和旧版本兼容，具体检索特性请参见相关章节。

## 5、位置描述符

|  |  |
| --- | --- |
| **符号** | **意义** |
| # | ’ STR1 # STR2’：表示包含STR1和STR2，且STR1、STR2在同一句中 |
| % | ’ STR1 % STR2’：表示包含STR1和STR2，且STR1 与STR2在同一句中, 且STR1在STR2前面 |
| /NEAR N | ’ STR1 /NEAR N STR2’：表示包含STR1和STR2，且STR1 与 STR2 在同一句中，且相隔不超过N个字词 |
| /PREV N | ’ STR1 /PREV N STR2’：表示包含STR1和STR2，且STR1 与 STR2 在同一句中，STR1在STR2前面不超过N个字词 |
| /AFT N | ’ STR1 /AFT N STR2’：表示包含STR1和STR2，且STR1 与 STR2 在同一句中，STR1在STR2后面且超过N个字词 |
| /SEN N | ’ STR1 /SEN N STR2’：表示包含STR1和STR2，且STR1 与 STR2 在同一段中，且这两个词所在句子的序号差不大于N。 |
| /PRG N | ’ STR1 /PRG N STR2’：表示包含STR1和STR2，且STR1 与 STR2 相隔不超过N段 |
| $ N | ‘STR $ N’：表示所查关键词STR最少出现N次 |

位置描述符主要用于TEXT、TEXTCHAR、LTEXT、LTEXTCHAR、TITLE、QTEXT等等文本索引类型，用来完成复杂的KSQL查询语句，需要注意的是前面所列位置描述符前后都必须至少包含一个空格用以分隔表达式中不同的部分，并且其中字母严格要求大写。

在不同的文本索引类型中，这些位置操作符，具体实现的意义会略有差别。

## 6、数据分隔符

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **编码** |  | **支持的分隔符** | | | | | |
| GBK | 半角字符 | , | ; | ! | \ |  |  |
| 全角字符 | ， | ； | ！ | 、 | ＠ | ￥ |
| UNICODE | 半角字符 | , | ; | ! | \ |  |  |
| 全角字符 | ， | ； | ！ | 、 | ＠ | ￥ |

数据分隔符主要用于多值型数据存储类型，即一条记录的单一字段可以有多个值，我们一般将这里的多个值称作子值。如MVCHAR、LMVCHAR类型就是KBase系统提供的两种多值型数据存储类型。

数据分隔符即是用在多个不同的值之间，用于分隔多个子值的分隔符。@＠

|  |
| --- |
| **注意：**系统中有一个配置选项，可以选择严格的数据分隔符，当选项启用时，数据分割符为半角的分号‘;’。 |

## 7、匹配运算符

|  |  |
| --- | --- |
| **符号** | **意义** |
| = | 匹配运算符 |

匹配运算符可用于各种索引类型中，不同的索引类型，匹配代表的意义不完全一致，具体请参阅数据索引类型相关章节。

## 8、模糊匹配运算符

|  |  |
| --- | --- |
| **符号** | **意义** |
| % | 模糊匹配运算符 |

模糊匹配运算符可用于各种索引类型中，不同的索引类型，模糊匹配代表的意义不完全一致，具体请参阅数据索引类型相关章节。

## 9、相关匹配运算符

|  |  |
| --- | --- |
| **符号** | **意义** |
| %= | 相关匹配运算符 |

KBase 10.0开始，支持相关匹配运算符，相关匹配运算符主要用在文本索引类型中，实现文本的相关查询，具体请参阅数据索引类型相关章节。

第三章 KSQL数据类型

KBase 8.0 开始，实现了数据类型的分离。不同于KBase 8.0以前的版本，在KBase 8.0中，数据类型由数据存储类型和数据索引类型来表达，存储类型和索引类型的组合，大大丰富了 KBase 系统能提供的数据特性。

为了保持与旧的系统兼容，原来的数据类型，作为一种预定义的数据组合类型保留，也即是，每种旧的数据类型——数据组合类型，相当于一种存储类型和索引类型的组合。

## 1、数据存储类型概述

KBase系统提供多种数据存储类型，以尽可能满足各种应用需求。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **存储类型** | **最大长度**  **(字节)** | **支持的编码** | **变长** | **说明** |
| INTEGER | 1-32(需指定) | ASCII | 否 | 整数，长度为用字符串形式保存数据的长度，数据范围是 32 位有符号整数的范围。 |
| DATE | 8-32(需指定) | 日期 |
| TIME | 14-32(需指定) | 时间 |
| NUM | 1-32(需指定) | 浮点数，长度为用字符串形式保存数据的长度 |
| INT64 | 14-32(需指定) | 整数，长度为用字符串形式保存数据的长度，数据范围是 64 位有符号整数的范围。 |
| AUTO | 12 | 自增，从1自动增加，不能指定值 总是有索引 |
| CHAR | 1-254(需指定) | ASCII、 UNICODE | 字符串，支持“单一”索引 |
| VCHAR | 254 | 是 | 变长字符串，支持“单一”索引 |
| MVCHAR | 32K | 多值字符串 |
| LMVCHAR | 64K/2G | 多值字符串 |
| MTEXT | 4K | 文本 |
| LTEXT | 16M | 文本 |
| VECTOR | 16M | ASCII | 向量字段 |
| MAP | \ | \ | \ | 虚字段，根据映射关系确定 |
| SOB | 254 | BIN | 是 | 二进制，不能索引 |
| LOB | 16M | 二进制，不能索引 |
| DOB | 2G | 数字对象字段 |
| DOCUMENT | 16M | ASCII | 文本 |
| DOCPATH | 1024 | 文本，保存的是连接到文档的文件名 |
| TNAME | \ | \ | \ | 虚拟字段，返回表名称 |
| TANAME | \ | \ | \ | 虚拟字段，返回表的显示名称 |
| RECORDID | \ | \ | \ | 虚拟字段，返回物理记录号 |
| RELEVANT | \ | \ | \ | 虚拟字段，返回检索结果的相关度 |
| SNAPSHOT | \ | \ | \ | 虚拟字段，返回与检索相关的特定字段的检索快照 |
| VIRTUAL | \ | \ | \ | 虚拟字段，返回其映射到的其他一个或多个字段 |

## 2、数据存储类型特性

### 2.1 INTEGER、NUM和INT64类型

INTEGER、INT64和NUM分别表示32位整数， 64位整数和浮点数，其可以表达的最大数据范围如下表。

|  |  |  |
| --- | --- | --- |
| **类型** | **最小值** | **最大值** |
| INTEGER | -231 +1 | 231-2 |
| INT64 | -263+1 | 263-2 |
| NUM | -3.4028233e+038 | 3.4028233e+038 |

与一般传统数据库不一样，KBase系统中的数值型数据存储类型的精度不仅受限于由其可表达的最大数据范围，还受限于其定义长度。这是因为KBase系统以字符的形式保存各种数值型数据，以便最大程度兼容各种不严格的数据表达形式。例如，如果INTEGER定义长度为4，则可能范围是（-999，9999），这时如果试图存入一个超出范围的数据，系统会自动进行截断，例如存入的数据是 -987654，则会截断为 -987，这样，你就不会得到想要的结果，因此一定要注意数据的定义及允许的数据范围。

NUM类型为浮点数类型，它有着比整数更大的数据范围，但在计算中会产生四舍五入的误差。KBase系统中，浮点数类型数据可以用科学记数法表达。

### 2.2 AUTO类型

数据类型AUTO为自增字段。一般其数据为从1开始随物理记录号递增的自然数。AUTO类型的特性是，其值在新增记录时可以由系统自动设定，并且，其值不可以修改。KBase 系统允许用户可以在增加记录时给自增数据设定特定的值，到要求这个特定的值不小于系统默认的值，如果用户指定的值小于系统默认的值，系统会忽略用户指定的值。

### 2.3 DATE 和 TIME类型

DATE和TIME是用于记录日期、时间的两种数据存储类型。

DATE类型表示日期，KBase系统的日期的由“年月日”表达，标准的日期格式有：

|  |  |
| --- | --- |
| 格式一 | yyyy/mm/dd |
| 格式二 | yyyy:mm:dd |
| 格式三 | yyyy-mm-dd |
| 格式四 | yyyymmdd |
| 格式五 | yyyy.mm.dd |
| 格式六 | yyyy年mm月dd日 |

其中：yyyy 表示年份，mm 表示月份，dd 表示日期

例如：2004/01/23,表示 2004年1月23日

|  |
| --- |
| **注意：**从KBase 11.0开始，日期的格式，还支持 mm-dd-yyyy 的格式，其中月份、日期、年份之间必须包含分割符，分割符可以是（‘:’，‘-’，‘/’，‘,’）。并且月份可以为英文月份，或英文形式的三字母缩写。 |

|  |  |
| --- | --- |
| 格式一： | hh:mm:ss |
| 格式二： | hh-mm-ss |
| 格式三： | hh/mm/ss |
| 格式四： | hhmmss |
| 格式五： | hh时mm分ss秒 |

TIME类型表示时间，KBase系统的时间有“年月日”和“时分秒”两部分组成，两部分之间可以用多种符号连接（‘ ’，‘:’，‘-’，‘/’，‘,’），“年月日”部分的标准格式同DATE，“时分秒”部分支持的标准格式如下：

其中：hh 表示时，mm 表示分，dd 表示秒

例如：2005-01-23:08-30-15，表示2005年1月23日8时30分15秒。

### 2.3 CHAR 和 VCHAR类型

CHAR类型用于存储定长字符串，其最大长度可由设计者自行设定，但最大长度不能超过254个字符。VCHAR类型用于存储变长字符串，但最大长度超过254个字符。

CHAR数据类型通常用于容纳如人名、公司名、地址等内容，因为这类数据的长度通常差别不大。例如，一般对保存文件名的字段，应将它定义为CHAR类型，这样，我们就可以对它进行精确匹配检索。

通常情况下，VCHAR会比CHAR更节省磁盘空间，尤其是在不同记录数据长度差别很大的时候。一般情况下，系统对VCHAR的存取相对于CHAR来说更费时。在数据最大长度比较小时，对CHAR有更高的存取速度，更优的缓存效率。如果数据最大长度小于16，总是应该选用CHAR类型，如果数据最大长度大于64，一般选择VCHAR会有更好的性能。

### 2.4 MVCHAR类型

MVCHAR是多值的CHAR类型。每个子值以多值分隔符分隔。KBase系统支持的多值分隔符有：

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 编码 |  | 支持的分隔符 | | | | | |
| GBK | 半角字符 | , | ; | ! | \ |  |  |
| 全角字符 | ， | ； | ！ | 、 | ＠ | ￥ |
| UNICODE | 半角字符 | , | ; | ! | \ |  |  |
| 全角字符 | ， | ； | ！ | 、 | ＠ | ￥ |

**注意：**系统中有一个配置选项STRICTLYDELIMITED，默认为OFF，支持以上全部多值分隔符；当该选项启用时，即设置为ON时，选择严格分隔符，数据分割符为半角的分号‘;’。建议新的应用系统，不管是否启用严格的数据分隔符，总是仅仅采用半角分号(;)作为数据分隔符。

目前，数据表也可以支持表级自定义多值字段的分隔符属性，优先级高于系统参数STRICTLYDELIMITED的配置。但需要注意的是，修改表级参数后，需要重建索引，自定义多值字段的分隔符属性才能生效。

**表级自定义分隔符设置方法：**

设置语句： ALTER TABLE <表名> ALTER <字段名> SET SPECPROPERTY <值>

参数说明：设置特别属性，以自定义分隔符：0-默认、1-全部分割符（10种）、2-严格分隔符（;）

不同参数值下分隔符选用情况对应下表：

|  |  |  |  |
| --- | --- | --- | --- |
| 系统参数STRICTLYDELIMITED | 表参数 SPECPROPERTY | | |
| 0 | 1 | 2 |
| ON | 严格分隔符 | 全部分隔符 | 严格分隔符 |
| OFF | 全部分隔符 | 全部分隔符 | 严格分隔符 |

可通过系统表(sys\_field)查看表级自定义分隔符属性设置。

### 2.5 LMVCHAR类型

LMVCHAR是另一种多值的CHAR类型。它与MVCHAR提供的应用级特性完全一样，区别在于系统底层的处理上，LMVCHAR更适合记录内容的反复修改。

### 2.6 MTEXT 和 LTEXT类型

MTEXT类型用于存储较小的文本，最大长度不能超过32K个字符。LTEXT类型用于存储较大的文本内容，最大长度可以达到4M个字符。

KBase系统利用MTEXT和LTEXT来储存各种文本数据，如期刊全文、报纸全文、图书全文等等。LTEXT实际存储的长度达到4M，这个数据长度对于实际的应用而言，几乎没有限制。这相当于一条记录单一字段容量达到2百万汉字，20万字图书都可以存放10本左右。

### 2.7 SOB、LOB和DOB类型

这三个字段用于处理无结构字节流，如存储压缩视频图像、音频数据、可执行代码等数据。SOB处理小对象，最大长度为254字节，LOB处理大对象，最大长度为4M字节，而DOB则有能力处理更大的对象，长度可达2G字节。这三种字段都没有索引，一般不用于检索。

SOB和LOB字段类型，其读取方法与其它字段一致，写入或修改必须是一次性的；而DOB字段则支持完全的流式读写。目前对这类字段只支持基于二次开发函数的数据操作。

### 2.8 VECTOR类型

VECTOR类型是KBASE系统提供的通用的向量模型，在多种场合有着广泛的应用。基于向量运算，我们可以利用它来做相似检索。

VECTOR表达格式为：W1 :F1 ,W2 :F2 ,…,Wn :Fn，其中Wi是特征词，Fi是对应的权重，权重设置的标准为权重平方和为 10000 的平方，在权重设置不合理的情况下，我们会等比例缩放到符合权重标准。

分隔符可选择半角逗号与半角分号，设置语法如下：

设置语句： ALTER TABLE <表名> ALTER <字段名> SET SPECPROPERTY <值>

参数说明：0-默认半角逗号分隔（,）、1-半角分号分隔（;）。需要注意的是，修改参数后，需要重建索引，分隔符属性才能生效。

### 2.9 DOCUMENT / DOCPATH类型

DOCUMENT和DOCPATH是KBase用于管理文档数据的数据存储类型。文档数据存储类型管理的详细说明参见第六章的《文档存储类型管理》。

### 2.10 VIRTUAL类型

KBase 系统开创性的提供了VIRTUAL类型，我们称之为虚字段。这种字段实际上并不占有物理存储空间，但是，一旦对这种字段定义了相应的映射关系，就可以在这种字段上进行各种实际的操作。系统支持的操作包括，读取内容、建立索引、进行检索等等。

我们可以对一个VIRTUAL类型数据定义一个或多个映射字段，具体映射操作见相关章节。如果对VIRTUAL字段定义了一个映射字段，那么你就完全可以对这个字段进行各种操作，如同操作被映射的字段一样，只是不能对数据内容进行直接的修改。如果对VIRTUAL字段定义了多个映射字段，读取内容时，我们得到的会是某一个被映射的字段的内容，这个字段是根据定义映射关系时设定的权重优先关系选出来的。

### 2.11 SNAPSHOT等类型

KBase 系统为了保持与旧版系统的兼容性，保留了TNAME、SNAPSHOT等一类函数型字段。新的应用中，应该尽量避免使用这些类型，而使用相应的函数。

|  |  |  |
| --- | --- | --- |
| **类型** | **意义** | **替换函数** |
| TNAME | 返回表名称 | GETSYSFIELD(‘\_ \_TABLENAME ’ ) |
| TANAME | 返回表的显示名称 | GETSYSFIELD(‘\_ \_TABLEALIASNAME’) |
| RECORDID | 返回物理记录号 | GETSYSFIELD(‘\_ \_RID’) |
| RELEVANT | 返回检索结果的相关度 | GETSYSFIELD(‘\_ \_RELEVANT’) |
| SNAPSHOT | 返回与检索相关的特定字段的检索快照 | GETSNAPSHOT ( ‘被映射的列名’ ) |

## 3、数据索引类型概述

KBase系统提供了丰富的数据索引类型，可以满足各种应用需求。KBase系统中数据索引类型分为四类，基本型数据索引类型、字符串数据索引类型、文本型数据索引类型、其它特定用途的数据索引类型。

* **数值型数据索引类型**

包括INTEGER、QINTEGER、NUM、DATE、QDATE、TIME、INT64，这几种数据类型特点与作用同传统关系数据库类似。

* **字符串数据索引类型**

包括CHAR、ECHAR、MVCHAR、EMVCHAR、WORD，这几种数据类型特点适用于不同应用目标的字符串型数据，提供不同的字符串检索。

* **文本型数据索引类型**

包括STRING、STRCHAR、MSTRCHAR、TEXT、TEXTCHAR、LTEXT、LTEXTCHAR、LFTEXT、TITLE、EXTITLE、EXTITLECHAR、QSTRING、QSTRCHAR、ABSTRACT、COMPACTQTEXT、QTEXT、TYTDTEXT、TYTEXT主要是为了管理非结构化文档而设计的。

这是KBase系统与传统关系数据库主要区别之一，借助于这些数据类型可以在KBase中实现高速简捷的全文检索。

* **其他特定用途的数据索引类型**

包括VECTOR、FINGERPRINT、SEARCHENGINE、MIX，这些字段类型各有特点，为KBase系统引入很多有特色的查询功能。

值得特别指出的是，上述字符串数据索引类型、文本型数据索引类型，都可以支持UNICODE数据存储类型的数据的索引。

对于各种数据存储类型，我们可以根据应用的需要选用各种索引类型，如下是各种数据存储类型通常情况下可以选用的索引类型对照表。

|  |  |  |
| --- | --- | --- |
| **存储类型** | **推荐的索引类型** | **可能合适的索引类型** |
| INTEGER | INTEGER、QINTEGER | INT64、NUM |
| DATE | DATE、QDATE |  |
| TIME | TIME |  |
| NUM | NUM |  |
| INT64 | INT64 | INTEGER 、NUM |
| AUTO | INTEGER |  |
| CHAR VCHAR | CHAR、ECHAR、STRING、STRCHAR、WORD、EXTITLE | 所有其他字符串、文本型索引类型 |
| MVCHAR LMVCHAR | MVCHAR、EMVCHAR、MSTRCHAR | 所有其他文本型索引类型 |
| VECTOR | VECTOR、SVSM、MVSM |  |
| MTEXT | EXTITLE、QSTRING、QTEXT | 所有其他文本型索引类型 |
| LTEXT | EXTITLE、QSTRING、QTEXT | FINGERPRINT、所有其他文本型索引类型 |
| VIRTUAL | SEARCHENGINE、MIX | 根据映射关系,所有其他类型 |
| SOB LOB DOB | 无 |  |
| DOCUMENT | EXTITLE、QSTRING、QTEXT | FINGERPRINT、所有其他文本型索引类型 |
| DOCPATH | EXTITLE、QSTRING、QTEXT | FINGERPRINT、所有其他文本型索引类型 |
| TNAME TANAME RECORDID RELEVANT SNAPSHOT | 无 |  |

## 4、数据索引类型特性

### 4.1 INTEGER、QINTEGER 类型

KBase 系统在INTEGER索引上支持的检索特性主要是为了实现整数这种数值的查询需求。

QINTEGER 检索特性与INTEGER完全一致，但QINTEGER通过额外的索引数据，使其“比较查询”的性能很高。当然，为了获取高性能，QINTEGER索引的数据量更大，在增加或修改数据时的代价更高。

需要特别注意的是，QINTEGER索引要求数据值的范围在（0，65536），超出此数据范围的记录将不被正常索引，导致可能相应记录不能正常检索到。实际应用中，数据值应该设计的尽量小，否则会致使索引代价太高。

KBase 系统在INTEGER索引上支持的检索特性包括：精确查询、比较查询、范围查询。

#### 4.1.1精确查询

检索语法： <检索字段名> = <检索数值>

检索结果： 返回字段值与“检索数值”相等的记录。

#### 4.1.2比较查询

检索语法： <检索字段名><比较运算符><检索数值>

检索结果： 返回字段值与“检索数值”相比，符合比较关系的记录。系统支持的比较运算符有： > 大于、< 小于、>= 大于等于、<= 小于等于。其比较方法是按照相应的数值大小来进行的。

如下面的KSQL查询语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 下载频次= ‘350’ |

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 下载频次>= ‘350’ |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **注意：**在KBase系统中，数据的表达式书写形式相对比较宽松。上述KSQL语句也可写为：   |  |  | | --- | --- | | **SELECT** | \* | | **FROM** | CCND2009 | | **WHERE** | 下载频次= 350 | |

实际上，几乎所有数据类型，都可以用单引号（’ Field Value’）引起来表达，也可以用双引号（”Field Value”）引起来表达，在不发生歧义的情况下，也可以不用引号。这里单引号、双引号都必须是ASCII，它们作为特殊符号出现，而不是中文全角的标点引号。

#### 4.1.3范围查询

检索语法： <检索字段名> BETWEEN ( <最小值>, <最大值>)

检索结果： 检索语义同：

( <检索字段名> >= <最小值> and <检索字段名> <= <最大值>)

一般情况下，BETWEEN比上述语法有更好的性能。

### 4.2 INT64类型

INT64检索功能特性与INTEGER完全一样。

### 4.3 NUM类型

按浮点数做索引，支持科学计数法。NUM支持的检索功能特性与INTEGER完全一样。

### 4.4 DATE、TIME和QDATE类型

KBase 系统在DATE、TIME索引上支持的检索特性主要是为了实现日期、时间这种数值的查询需求，DATE对应日期类型，TIME时间上对应的是时间戳类型。

QDATE 检索特性与DATE完全一致，但QDATE通过额外的索引数据，使其“比较查询”的性能很高。当然，为了获取高性能，QDATE索引的数据量更大，在增加或修改数据时的代价更高。

KBase 系统在这几个索引上支持的检索特性包括：精确查询、比较查询、模糊查询、范围查询。

#### 4.4.1精确查询

检索语法： <检索字段名> = <检索值>

检索结果： 返回字段值与“检索值”相等的记录。

#### 4.4.2比较查询

检索语法： <检索字段名><比较运算符><检索值>

检索结果： 返回字段值与“检索值”相比，符合比较关系的记录。系统支持的比较运算符有： > 大于、< 小于、>= 大于等于、<= 小于等于。其比较方法是按照相应的数值大小来进行的。

#### 4.4.3模糊查询

检索语法： <检索字段名> % <检索值>

检索结果： 返回字段值与“检索值”模糊匹配的记录。对时间和日期，KBase系统模糊匹配的方法是通过将<检索值>转化为<检索范围值>，再通过范围查找来实现。

示例1：查找“CCND2009”表中“发表时间”为“2009年05月01日”的记录，利用精确查询可以用 如下 SQL 语句实现。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 发表时间=”2009-05-01” |

示例2：查找“CCND2009”表中“发表时间”为2009年05月的记录。利用比较查询可以用 如下 SQL 语句实现。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 发表时间>=”2009-05-01” |
| **AND** | 发表时间<”2009-06-01” |

对于示例2，我们也可以利用模糊查询用 如下 SQL 语句实现。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 发表时间%”2009-05” |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **注意：**如果提交如下检索语句：   |  |  | | --- | --- | | **SELECT** | \* | | **FROM** | CCND2009 | | **WHERE** | 发表时间=”2009-05” |   由于”2009-05” 是一个时间范围，其检索含义会自动转换为：   |  |  | | --- | --- | | **SELECT** | \* | | **FROM** | CCND2009 | | **WHERE** | 发表时间%”2009-05” | |
|  |

#### 4.4.4范围查询

检索语法： <检索字段名> BETWEEN ( <最小值>, <最大值>)

检索结果： 检索语义同：

( <检索字段名>>= <最小值> and <检索字段名>>= <最大值>)

一般情况下，BETWEEN比上述语法有更好的性能。

### 4.5 CHAR和 ECHAR类型

KBase 系统在CHAR、ECHAR索引上支持的检索特性比较丰富。ECHAR与CHAR具有完全相同的检索特性，只是ECHAR更适合索引“性别”这种有大量重复数据的列。

#### 4.5.1 精确查询

检索语法： <检索字段名> = 检索词

检索结果： 返回字段值与“检索词”完全匹配（但忽略字符大小写）的记录

注意：“检索词”只支持前128字节索引

例如，检索“CCND2009”表的“文件名”列为“SZSB20020514A011”的记录，对应的KSQL查询语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 文件名='SZSB20020514A011' |

在KBase系统中，所有检索值都是大小不敏感的，因此如下SQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 文件名='szsb20020514A011' |

和上面的语句是等价的，查询结果一致。

#### 4.5.2前方一致查找

检索语法： <检索字段名> = 检索词 + <\*/?>

检索结果： 返回字段值前面部分与“检索词”完全匹配（但忽略字符大小写）的记录。

建议：建议使用‘\*’ 匹配操作符，这样与‘\*/?’模式匹配检索语义完全一致，保留‘?’ 匹配操作符是为了与旧版系统兼容。

|  |
| --- |
| **注意：**当使用‘?’匹配操作符时，检索词中，不可出现？或\*匹配操作符，否则检索语义发生变化，这时不是按照前方一致进行检索了，而是进行的‘\*/?’模式匹配检索，具体见后面的说明。另外，如果检索词首字符是‘\*’或‘?’，这样检索语义有歧义，检索不会有结果。 |

例如，检索“CCND2009”表的“文件名”列中，以字符串“SZSB”开始的所有记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 文件名='SZSB\*' |

将能命中“文件名”为“SZSB20020514A011”的记录。

#### 4.5.3包含查找

检索语法： <检索字段名> = <\*/?> + 检索词

检索结果： 返回字段值中包含“检索词”（但忽略字符大小写）的记录。

|  |
| --- |
| **注意：**这种检索执行效率较低，不建议使用这种语法，如果应用中大量需要这种查询，可以考虑采用其他如 WORD、STRCHAR 等索引类型。并且，这样使得检索词中，不可出现‘\*’或‘?’匹配操作符，否则一般不会有结果，因为‘\*’或‘?’与模式比较控制符冲突。  建议：同样建议使用‘\*’ 匹配操作符，这样与‘\*/?’模式匹配检索语义完全一致，保留‘?’ 匹配操作符是为了与旧版系统兼容。 |

例如，检索“CCND2009”表的“文件名”列中，包含字符串’ 2002’的所有记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 文件名='\*2002' |

将也能命中“文件名”为“SZSB20020514A011”的记录。

#### 4.5.4 ?/\* 模式查找

检索语法： <检索字段名> = 检索模式串

检索结果： 返回与“检索模式串”完全匹配的记录。

检索模式串包含‘\*’或‘?’，‘\*’表示匹配任意个字符，‘?’表示匹配1个字符，英文字母忽略大小写匹配。需要注意的是，检索模式串的首字符不能是‘\*’或‘?’匹配符，原因是这样与**包含查找**语义冲突。此时，其检索语义与 WORD 字段类型的语义完全一致。具体功能可以参见 WORD 的说明。

另外，这里字符的意义同GB18030标准，而不是指字节。一个ASCII是一个字符（占一字节），一个汉字也是一个字符（占两字节）。

例如，检索“CCND2009”表的“文件名”列中，以字符串’ SZSB’开头，并且’ SZSB’ 和’ A’ 之间相隔4个或4个以上个字符，子串’A’后有任意个字符记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 文件名='SZSB????\*A\*' |

#### 4.5.5 比较查找

检索语法： <检索字段名><比较运算符><检索词>

检索结果： 返回字段值与“检索词”相比，符合比较关系的记录。系统支持的比较运算符有： > 大于、< 小于、>= 大于等于、<= 小于等于。其比较方法是按照忽略大小写的字符串大小比较来进行的。

例如，检索“CCND2009”表的“文件名”列中，查出大于等于子串’ SZSB20020514A1’的记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 文件名>='SZSB20020514A1' |

|  |
| --- |
| **注意：**鉴于KBase系统是不区分大小写的，因此，字符串的比较是忽略大小写的。 |

#### 4.5.6 模糊查找

检索语法： <检索字段名> % <检索词>

检索结果： 返回字段值与“检索词”相比，符合模糊匹配关系的记录。对于CHAR系列的索引，系统支持的模糊匹配算法类似于反向的前方一致查询，即记录对应值与“检索词”前面部分一致。这里的一致同样是忽略大小写的字符比较。

例如，检索“CCND2009”表的“分类号”列中，查出与“E12”匹配或与前面部分串匹配的记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 分类号% 'E12' |

这个SQL将能命中分类号为“E12”、“E1”、“E”的记录，注意，不能命中分类号为“E12\_03”等形式的记录。

### 4.6 MVCHAR和 EMVCHAR类型

MVCHAR就是多值的CHAR索引，即一个MVCHAR字段，由多个CHAR组成，每个CHAR为一个子值，子值由分隔符分隔。MVCHAR索引类型能够识别的子值分隔符同数据存储类型的MVCHAR一致。

同样，EMVCHAR与MVCHAR具有完全相同的检索特性，只是EMVCHAR更适合索引具有大量重复数据的列。

所有CHAR类型支持的检索特性，MVCHAR和EMVCHAR都支持。

例如，检索“CCND2009”表的“专题代码”列中，查出含有子值“F092”的记录。相应的KSQL查询语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 专题代码='F092' |

需要注意的是，如果想查询“专题代码”既为'E072'又为'E073'的记录，如果用如下KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 专题代码='E072;E073;' |

是检索不到所需结果的。因为系统会认为你是在查一个值为'E072;E073;'的专题代码。正确的SQL语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 专题代码='E072' |
| **AND** | 专题代码='E073' |

到目前为止，系统还没有限定一个MVCHAR类型最多可以索引多少个子值，只限定了可以索引的数据总长度，最大支持32K 字节。这32K 字节包含所有值及其分隔的总长。

### 4.7 WORD类型

KSQL支持对WORD数据类型?和\*两种匹配检索。其中? 代表一个字符，\* 代表任意个字符。查找时，? 和\* 可以出现在检索条件的任意位置，出现任意次。

#### 4.7.1 精确查询

同 CHAR 索引一样，支持精确查询。

具体内容参见CHAR索引的说明。

注意：“检索词”只支持前64字节索引，只要前64字符完全匹配，则命中，超出64字符的截断后再匹配。

WORD数据类型不同于CHAR类型，不支持比较查询。

#### 4.7.2 ?/\* 模式查找

检索语法： <检索字段名> = 检索模式串

检索结果： 返回与“检索模式串”完全匹配的记录。

检索模式串包含‘\*’或‘?’，‘\*’表示匹配任意个字符，‘?’表示匹配1个字符，英文字母忽略大小写匹配。

这里字符的意义同GB18030标准，而不是指字节。一个ASCII是一个字符（占一字节），一个汉字也是一个字符（占两字节）。

WORD字段类型支持‘\*’或‘?’匹配符的任意组合查找，检索模式串中的‘\*’或‘?’可以以任意方式组合，出现在任何位置。

例如一，检索“CCND2009”表的“题名”列中，以“城市”和 “发展”之间相隔3个或3个以上字符，且以“城市”开始，“发展”结尾的记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 题名='城市\*???发展' |

例如二，检索“CCND2009”表的“题名”列中，以“城市”和“道路”之间相隔任意个字符，“道路”和“发展”之间相隔1个字符，且以“城市”开始，“发展”结尾的记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 题名='城市\*道路?发展' |

### 4.8 STRING和 STRCHAR类型

STRING和STRCHAR是KBase系统中最小型的索引文本数据的类型，它仅仅针对数据的前面254字节做索引。

STRING和STRCHAR区别是，STRING在做索引时，是分词后索引的。而STRCHAR是按字切分索引的，“按字切分索引”我们通常称之为不分词索引。

这两种类型具有文本型数据分词索引的特点，与其它文本型数据类型的区别是，它们不分句，更不分段，即它们仅相当于其它文本型数据中的一个句子。因此，如果一个字段只有少量文本数据，采用STRING或STRCHAR，将可能获得更佳的检索性能。

这两种类型与其它文本型数据类型相比，还有一个细微的差异是，STRING或STRCHAR在分词（或分字）后，直接完全索引；而其它文本型数据类型则是在分词（或分字）后，先过滤了停用字词之后，再进行索引。这一点，对检索有时会有一些微妙的影响。关于停用词更详细的说明，可参考后面TEXT字段类型的说明。

#### 4.8.1精确查找

检索语法： <检索字段名> =<检索词>

检索结果： 返回字段值中包含“检索词”（但忽略字符大小写）的记录。

精确查找实际上是短语包含查找，这是最基本的文本型数据的检索方式。与CHAR不同，文本型数据的包含查找不用加上问号（?/\*）作内容前缀。

因为STRING类型的索引是基于分词的，查找的内容应是一个或多个连续的词汇，如果要查找的内容不是一个准确的概念，仅是其中的一个子串，也可能无法检索到。

例如：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2003 |
| **WHERE** | 篇名='农业科' |

上面的ＫSQL不能命中“篇名”为“近代美国农业科技的引进及其影响评述”的记录，但可以命中“篇名”为“农业科宣传记事”的记录。这是因为“近代美国农业科技的引进及其影响评述”索引时是先分词为“近代/美国/农业/科技/的/引进/及其/影响/评述”，这样，我们只可检索其中的一个或多个连续的词。通过分词索引，我们可以提高文本检索的查准率，同时可能会降低查全率。

如下的ＫSQL，则可以命中“篇名”为“近代美国农业科技的引进及其影响评述”的记录。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2003 |
| **WHERE** | 篇名='农业科技' |

#### 4.8.2 ?/\* 模式查找

检索语法： <检索字段名> = <检索词位置控制表达式>

<检索词位置控制表达式> 以位置描述标记（\*和?）分隔两个子串STR1和STR2组合成，有两种格式：

|  |  |
| --- | --- |
| **格式** | **含义** |
| “STR1\*STR2” | 表示STR1与STR2相隔任意个字符，STR1在前； |
| “STR1?…?STR2” | 表示STR1与STR2相隔0到n个字符，n等于表达式中? 号的个数，STR1在前； |

检索结果： 返回字段值必须包含STR1和STR2，并且STR1在STR2前面，相隔不超过一定数量的字或词的记录。

例如：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2003 |
| **WHERE** | 篇名='债务\*经济' |

将能查出“篇名”字段中，包含子串'债务'和'经济'，'债务'和'经济'是词, '债务'和'经济'之间相隔任意个字符串，且子串'债务'在'经济'前面的记录。比如可以命中篇名为“从债务看商品经济状况”的记录。

|  |
| --- |
| **注意**，STRING或STRCHAR的模式查找不同于CHAR或WORD，区别在于：   1. STRING的语法不要求命中以 “STR1”开头的记录。 2. 用问号（？）表达的距离以词计算，并且是不超过这个距离即为命中。 |
| 另外，如下两条语句意义相同，检索结果一致。   |  |  | | --- | --- | | **SELECT** | \* | | **FROM** | CJFD2003 | | **WHERE** | 篇名='经济' \* '债务' |  |  |  | | --- | --- | | **SELECT** | \* | | **FROM** | CJFD2003 | | **WHERE** | 篇名=经济' AND 篇名= '债务' | |

因此，上面的KSQL 查询实际上并不是**位置查找**，而是一个逻辑组合条件的查询语句，即两个**包含查找**的AND运算。有关运算符\*号的使用规则请参考KSQL语法的说明。

为了减少类似问题，建议在书写SQL词句时，请尽量用引号将检索关键字引起来。

### 4.9 MSTRCHAR类型

MSTRCHAR是KBASE系统提供的另一种多值类型索引。可以简单的看作MVCHAR和STRCHAR类型的结合体，它和MVCHAR一样存储多值，但它提供了更强的检索功能，尤其是MSTRCHAR支持类似于数组的子值序位检索。

#### 4.9.1 精确查询

检索语法： <检索字段名> = <检索词>

检索结果： 返回字段值与“检索词”完全匹配（但忽略字符大小写）的记录

例如，检索“CCND2009”表的“作者”列为“梅承恩”的记录，对应的KSQL查询语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 作者='梅承恩' |

#### 4.9.2 模糊查找

检索语法： <检索字段名> % <检索词>

检索结果： 返回字段值与“检索词”相比，符合模糊匹配关系的记录。对于MSTRCHAR索引，系统支持的模糊匹配算法是包含，即记录对应值包含“检索词”计算命中。

例如，检索“CCND2009”表的“作者”列含有“李”的记录，对应的KSQL查询语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 作者%'李' |

这个KSQL将会命中姓“李”的作者，当然，也会命中“小李子”这样的作者名。

#### 4.9.3序位查询

检索语法： <检索字段名> =|% <检索词序位控制表达式>

<检索词序位控制表达式> 格式： “检索词 /SUB N”，其中，“/SUB N”是序位检索控制符，N表示要在第N个子值中进行查找。

检索结果： 返回字段的第N个子值与“检索词”相比，符合精确（=）或模糊（%）匹配关系的记录。

例如，检索“期刊”表的“作者”列的第一个子值为“梅承恩”的记录，对应的KSQL查询语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 作者='梅承恩 /SUB 1' |

这个KSQL将会命中第一作者是“梅承恩”的记录。

### 4.10 TEXT、TEXTCHAR、LTEXT和LTEXTCHAR类型

TEXT和TEXTCHAR是KBase系统中最早提供的大文本数据索引类型，在最初的版本中就已经出现。TEXT与TEXTCHAR索引提供了传统图书情报专业教科书中提到的各种全文检索特性。

LTEXT和LTEXTCHAR是TEXT和TEXTCHAR的改进索引类型。新的系统中，应尽可能采用这些字段类型，其差别体现在性能上。改进后的LTEXT对比TEXT，在检索速度有显著提高，并且索引的数据量节省了25%左右。

TEXT、LTEXT是分词索引的，而TEXTCHAR、LTEXTCHAR是不分词索引的。一般全文检索时，有两个应用级的关键指标：查全率和查准率。理论上不分词能做到100%的查全率，但远不能达到人们查准的要求，特别是在海量文献中，不分词往往找出一大堆仅是匹配上查询关键词但与所查概念完全无关的数据。实践证明，分词对提高查准率有很好的帮助。如：“原子组成分子”，如果不分词，当查询“成分”时，就会返回这条数据，如果选择分词类型字段，因它在索引中表现为“原子/组成/分子”，因此就不会返回这条与“成分”不相关的数据。不分词的字段字型往往用在一些如机构名称、人名、期刊名等需要通过只言片语返回一些线索以进一步查询的内容上。

TEXT系列全文索引支持的检索特性非常丰富，主要包括：

#### 4.10.1精确查找

检索语法： <检索字段名> = <检索词>

检索结果： 返回字段值与“检索词”完全匹配（但忽略字符大小写）的记录

TEXT语法格式及功能特点和前面的STRING索引类型的精确查找一样，但要求检索词必须出现在一句中。所有基于关键词检索的全文检索系统，都会提供这种检索特性。

#### 4.10.2位置查找

检索语法： <检索字段名> = <检索词位置控制表达式>

检索结果： 返回字段值与<检索词位置控制表达式>匹配的记录。

<检索词位置控制表达式>位置描述标记及其匹配方法列表如下：

|  |  |
| --- | --- |
| **格式** | **含义** |
| ‘STR1**#** STR2’ | 表示包含STR1和STR2，且STR1、STR2在同一句中； |
| ‘STR1 **%** STR2’ | 表示包含STR1和STR2，且STR1 与STR2在同一句中, 且STR1在STR2前面； |
| ‘STR1 **/NEAR N** STR2’ | 表示包含STR1和STR2，且STR1 与 STR2 在同一句中，且相隔不超过N个词/字； |
| ‘STR1 **/PREV N** STR2’ | 表示包含STR1和STR2，且STR1 与 STR2 在同一句中，STR1在STR2前面不超过N个词/字； |
| ‘STR1 **/AFT N** STR2’ | 表示包含STR1和STR2，且STR1 与 STR2 在同一句中，STR2在STR1后面至少N个词/字； |
| ‘STR1 **/SEN N** STR2’ | 表示包含STR1和STR2，且STR1 与 STR2 在同一段中，且这两个词所在句子的序号差不大于N； |
| ‘STR1 **/PRG N** STR2’ | 表示包含STR1和STR2，且这两个词所在段落的序号差不大于N。 |

但对比STRING字段类型，TEXT等文本索引类型提供了更丰富的位置查找方法。对于几千字几万字的内容，是很有必要进行更准确的位置查找。其技术上的内在原因是，TEXT等大文本字段在作索引时，不仅记录了每个词在句中的位置序号，还记录了词所在的段落序号，句子序号。而STRING字段类型总是作为单一句子来处理。

|  |
| --- |
| **注意：**前面所列位置描述标记前后都必须至少包含一个半角空格用以分隔表达式中不同的部分。 |

例如1，检索“CCND2009”表的“全文”列中，包含子串'网格'和'系统'，且'网格'和'系统'出现在同一句中的记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 全文='网格 # 系统' |

例如2，检索“CCND2009”表的“全文”列中，含子串'信息'和'系统'，子串'信息'和'系统'出现在同一句中，'信息'出现在'系统'前面，且相隔不超过4个词（TEXTCHAR字段为字）的记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 全文='信息 /PREV 4 系统' |

例如2，检索“CCND2009”表的“全文”列中，包含子串'解决'和'战争'，'解决'和'战争'出现在同一段中，且相隔不超过3个句子（4-1个句子）的记录。KSQL语句：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 全文='解决 /SEN 4 战争' |

#### 4.10.3词频查找

检索语法： <检索字段名> = “检索词 **$ N**“

其中，N表示所查关键词STR最少出现的次数。

检索结果： 返回字段值中包含<检索词>，并且最少出现N次的记录。

有时候，查询人员在查询一个关键词时，可能要求这个关键词在记录中出现很多次，他才认为这条记录是他想要的。这时可以借助于词频查找功能。

例如，我们在表CCND2009库中检索’计算机’这个词在“正标题”中至少出现两次的记录的结果。检索所用的KSQL语句是：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 正标题='计算机 $ 2' |

#### 4.10.4自动扩展查询

检索语法： <检索字段名> = XLS（ “检索词 “[,扩展词典]）

检索结果： 返回字段值包含<检索词>或者<检索词>的扩展词的记录。

自动扩检是利用KBase系统自带的扩展检索词典（SYS\_XLDICT）来实现的，如果检索词在扩展检索词典中有对应的词条，那么，KBase系统会利用扩展出的词条进行自动扩检。

例如检索所用的KSQL语句是：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 正标题='计算机 $ 2' |

如果扩展检索词典（SYS\_XLDICT）的“计算机”对应的扩展词条有：“电脑”、“computer”，那么实际上相当于我们执行了如下检索：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CCND2009 |
| **WHERE** | 正标题='计算机' |
| **OR** | 正标题='电脑' |
| **OR** | 正标题=' computer ' |

|  |
| --- |
| 需要注意的是，自动扩展检索可以用在前面提到的所有检索模式上，包括：包含检索、位置检索、词频查找。 |

#### 4.10.5 模糊查找

检索语法： <检索字段名> % <检索词>

检索结果： 返回字段值与“检索词”相比，符合模糊匹配关系的记录。

对于TEXT系列索引，系统支持的模糊匹配算法是将检索词进行切词，如果这些词全部出现在对应的文本中，就算命中，而不同于精确查找，要求检索词相邻并出现在同一句中。

自动扩展查询也提供模糊查找的方式。

### 4.11 TITLE、EXTITLE、EXTITLECHAR、QSTRING、QSTRCHAR类型

TITLE索引类型，是为了专门适应针对文章标题这种数据来设计的。其支持的检索与LTEXT完全一样。

TITLE特别之处是索引时不分句、不分段，也就是说全部文本只是一个句子，因此，对于句子、段落的位置检索实际上与同句的位置检索一致。

EXTITLE基本同TITLE，其区别是TITLE索引时采用了停用字，EXTITLE索引时不采用停用字，在实际应用中可能更适合文章标题这种数据类型。

EXTITLECHAR 与 EXTITLE 检索特性一致，区别仅仅在分词处理上，也就是说，EXTITLE支持分词，而EXTITLECHAR 不分词，按字符来做索引。

KBase 10.0 开始支持相关查询，为了有更好的相关查询效果，开发了EXTITLE的升级版QSTRING，两者在功能特性方面完全一致。同样，QSTRCHAR是EXTITLECHAR的升级版本，也仅仅是为了优化相关查询效果。

这些索引支持的检索特性主要包括：

#### 4.11.1精确查找

检索语法： <检索字段名> = <检索词>

检索结果： 返回字段值与“检索词”完全匹配（但忽略字符大小写）的记录

#### 4.11.2位置查找

检索语法： <检索字段名> = <检索词位置控制表达式>

检索结果： 返回字段值与<检索词位置控制表达式>匹配的记录。

<检索词位置控制表达式>位置描述标记及其匹配方法列表如下：

|  |  |
| --- | --- |
| **格式** | **含义** |
| ‘STR1**#** STR2’ | 表示包含STR1和STR2 |
| ‘STR1 **%** STR2’ | 表示包含STR1和STR2， 且STR1在STR2前面； |
| ‘STR1 **/NEAR N** STR2’ | 表示包含STR1和STR2，且相隔不超过N个词/字； |
| ‘STR1 **/PREV N** STR2’ | 表示包含STR1和STR2，STR1在STR2前面不超过N个词/字； |
| ‘STR1 **/AFT N** STR2’ | 表示包含STR1和STR2，STR2在STR1后面至少N个词/字； |
| ‘STR1 **/SEN N** STR2’ | 同‘STR1**#** STR2’，为了保留形式上的支持 |
| ‘STR1 **/PRG N** STR2’ | 同‘STR1**#** STR2’， 为了保留形式上的支持 |

#### 4.11.3词频查找

检索语法： <检索字段名> = “检索词 **$ N**“

其中，N表示所查关键词STR最少出现的次数。

检索结果： 返回字段值中包含<检索词>，并且最少出现N次的记录。

#### 4.11.4自动扩展查询

检索语法： <检索字段名> = XLS（ “检索词 “[,扩展词典]）

检索结果： 返回字段值包含<检索词>或者<检索词>的扩展词的记录。

#### 4.11.5 模糊查找

检索语法： <检索字段名> % <检索词>

检索结果： 返回字段值与“检索词”相比，符合模糊匹配关系的记录。

#### 4.11.6全文索引中停用词的影响

TITLE 和EXTITLE的区别在乎停用词的处理。在各种全文索引方法中，一般对于各种分词的索引，我们一般会引入停用词，包括TITLE，但对于EXTITLE、QSTRING是不使用停用词的。

例如，表 期刊 字段“篇名”，其五条记录如下

|  |  |
| --- | --- |
| **记录序号** | **“篇名”字段值** |
| 1 | 地对空导弹 |
| 2 | 空对地导弹 |
| 3 | 对空扫描 |

执行KSQL：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | 期刊 |
| **WHERE** | 篇名=“地对空” |

如果字段“篇名”为 LTEXT、COMPACTQTEXT、QTEXT、TITLE等索引类型，则能命中记录序号为 1、3 的记录。

如果字段“篇名”为 ABSTRACT索引类型，则能命中记录序号为 1、2、3 的记录。

如果字段“篇名”为 EXTITLE、QSTRING索引类型，则能准确命中记录序号为 1 的记录。

### 4.12 ABSTRACT类型

ABSTRACT索引类型，是为了提高全文检索性能来实现的。其支持的检索功能与TEXT完全一样。

这两者索引提高性能的方法主要是通过索引的裁剪来实现的。特点是索引时系统对每个词所在位置信息只记录到句子序号。因此，对于“精确检索”、“位置检索”的实现上，无法实现与TEXT字段一样精确支持。虽然“精确检索”的效果有所下降，但这种检索更接近于当前索引引擎的默认检索方式，但在大多数情况下，仍然有很好的检索效果。

ABSTRACT索引支持的检索特性主要包括：

#### 4.12.1精确查找

检索语法： <检索字段名> = <检索词>

检索结果： 返回字段值与“检索词”“精确”匹配的记录

ABSTRACT由于索引信息中只记录到句子序号，“精确”匹配的方法受到索引结构的限制。例如查询检索词“检索功能”和“功能检索”得到的结果会完全一样。

#### 4.12.2位置查找

检索语法： <检索字段名> = <检索词位置控制表达式>

检索结果： 返回字段值与<检索词位置控制表达式>匹配的记录。

<检索词位置控制表达式>位置描述标记及其匹配方法列表如下：

|  |  |
| --- | --- |
| **格式** | **含义** |
| ‘STR1**#** STR2’ | 表示包含STR1和STR2，且STR1、STR2在同一句中； |
| ‘STR1 **%** STR2’ | 同‘STR1**#** STR2’， 为了保留形式上的支持 |
| ‘STR1 **/NEAR N** STR2’ | 同‘STR1**#** STR2’， 为了保留形式上的支持 |
| ‘STR1 **/PREV N** STR2’ | 同‘STR1**#** STR2’， 为了保留形式上的支持 |
| ‘STR1 **/AFT N** STR2’ | 同‘STR1**#** STR2’， 为了保留形式上的支持 |
| ‘STR1 **/SEN N** STR2’ | 表示包含STR1和STR2，且这两个词所在句子的序号差不大于N； |
| ‘STR1 **/PRG N** STR2’ | 表示包含STR1和STR2，且相隔不超过16\*（N+1）个句子； |

另外，需要注意的是，ABSTRACT类型的分句算法与TEXT、LTEXT有所不同，ABSTRACT分句算法更接近于自然语言的分句。

#### 4.12.3词频查找

检索语法： <检索字段名> = “检索词 **$ N**“

其中，N表示所查关键词STR最少出现的次数。

检索结果： 返回字段值中包含<检索词>，并且最少出现N次的记录。

#### 4.12.4自动扩展查询

检索语法： <检索字段名> = XLS（ “检索词 “[,扩展词典]）

检索结果： 返回字段值包含<检索词>或者<检索词>的扩展词的记录。

#### 4.12.5 模糊查找

检索语法： <检索字段名> % <检索词>

检索结果： 返回字段值与“检索词”相比，符合模糊匹配关系的记录。

### 4.13 COMPACTQTEXT类型

COMPACTQTEXT和ABSTRACT各种特性几乎一样，但由于 COMPACTQTEXT 索引记录了分词后的常用组合词的信息，因此一方面不仅提高了检索的性能，另一方面也改善了ABSTRACT的“精确检索”效果。

例如，如果按照ABSTRACT来索引，即使用户想精确检索“特别之处是TITLE不分段不分句”，也能检索到“TITLE特别之处是不分句、不分段”对应的记录。如果按照COMPACTQTEXT来索引，实现的就更加精确，上面的检索结果就不会出现。

COMPACTQTEXT具体用法和功能请参阅ABSTRACT。

### 4.14 QTEXT类型

QTEXT索引类型，是一种兼顾检索性能和精确检索效果的索引结构。其检索特性与LTEXT几乎完全一样。

QTEXT检索性能优异，是TEXT、LTEXT、ABSTRACT、COMPACTQTEXT的升级版本。

QTEXT索引支持的检索特性主要包括：

#### 4.14.1精确查找

检索语法： <检索字段名> = <检索词>

检索结果： 返回字段值与“检索词”完全匹配（但忽略字符大小写）的记录

#### 4.14.2位置查找

检索语法： <检索字段名> = <检索词位置控制表达式>

检索结果： 返回字段值与<检索词位置控制表达式>匹配的记录。

<检索词位置控制表达式>位置描述标记及其匹配方法列表如下：

|  |  |
| --- | --- |
| **格式** | **含义** |
| ‘STR1**#** STR2’ | 表示包含STR1和STR2，且STR1、STR2在同一句中； |
| ‘STR1 **%** STR2’ | 表示包含STR1和STR2，且STR1 与STR2在同一句中, 且STR1在STR2前面； |
| ‘STR1 **/NEAR N** STR2’ | 表示包含STR1和STR2，且STR1 与 STR2 在同一句中，且相隔不超过N个词/字； |
| ‘STR1 **/PREV N** STR2’ | 表示包含STR1和STR2，且STR1 与 STR2 在同一句中，STR1在STR2前面不超过N个词/字； |
| ‘STR1 **/AFT N** STR2’ | 表示包含STR1和STR2，且STR1 与 STR2 在同一句中，STR2在STR1后面至少N个词/字； |
| ‘STR1 **/SEN N** STR2’ | 表示包含STR1和STR2，且这两个词所在句子的序号差不大于N； |
| ‘STR1 **/PRG N** STR2’ | 表示包含STR1和STR2，且相隔不超过16\*（N+1）个句子； |

#### 4.14.3词频查找

检索语法： <检索字段名> = “检索词 **$ N**“

其中，N表示所查关键词STR最少出现的次数。

检索结果： 返回字段值中包含<检索词>，并且最少出现N次的记录。

#### 4.14.4自动扩展查询

检索语法： <检索字段名> = XLS（ “检索词 “[,扩展词典]）

检索结果： 返回字段值包含<检索词>或者<检索词>的扩展词的记录。

#### 4.14.5 模糊查找

检索语法： <检索字段名> % <检索词>

检索结果： 返回字段值与“检索词”相比，符合模糊匹配关系的记录。

#### 4.14.6 相关查找

检索语法： <检索字段名> %= <检索词>

检索结果： 返回字段值与“检索词”相比，最相关的部分记录。

### 4.15 LFTEXT类型

其功能检索特性与QTEXT几乎完全一样，差别在于LFTEXT仅仅使用回车和换行符作为分句符号。

LFTEXT具体用法和功能请参阅QTEXT。

### 4.16 TYTDTEXT和TYTEXT类型

基本检索功能与LTEXTCHAR完全一样。

TYTDTEXT和TYTEXT特点在于，在匹配的时候，汉字按照同音字来匹配，而不同于其他全文索引类型，匹配方式是按照汉字内码来进行的。其中 TYTDTEXT 匹配同音字并且区分音调，TYTEXT 则不区分音调。需要注意的是，对于多音的汉字，以其常用音为准。

GBK 字符集中，将繁体字和简体字统一到这一个编码字符集中。同一个字不同字形，发音当然一致。所以，同音字也支持繁简统一。

在最新的系统中，提供了最新的索引模式，“词干”索引，支持汉字繁简统一、英文词干统一。下面以示例说明这其中的差别：

表 TEST 字段“词汇”，其五条记录如下

|  |  |
| --- | --- |
| **记录序号** | **“词汇”字段值** |
| 1 | 实施 |
| 2 | 试试 |
| 3 | 石狮 |
| 4 | 事实 |
| 5 | 實施 |

执行KSQL：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | test |
| **WHERE** | 词汇=实施 |

如果字段“词汇”为 LTEXTCHAR，“正常”索引模式，则能命中记录序号为 1 的记录。

如果字段“词汇”为 LTEXTCHAR，“词干”索引模式，则能命中记录序号为 1、5 的记录。

如果字段“词汇”为 TYTDTEXT，“正常”索引模式，则能命中记录序号为 1、3、5 的记录。

如果字段“词汇”为 TYTEXT，“正常”索引模式，则能命中记录序号为 1、2、3、4、5 全部记录。

### 4.17 VECTOR、SVSM、MVSM类型

VECTOR是KBase系统提供的基于向量运算的索引类型，为了有更高效的查询效果，权重过低的部分结果记录会被舍弃，不参与计算。

SVSM类型与VECTOR类型的区别，在于SVSM类型所有命中的结果记录号都会参与计算，性能上有所差异。MVSM是VECTOR的简化版。

这些索引类型支持的检索语法：

检索语法： <检索字段名> = [参数:N<,|;>]<检索词>

其中：

<检索词>:: =<W1:F1,W2:F2,…Wi:Fi> 或者 <W1:F1;W2:F2; …Wi:Fi>

[参数:N<,|;>]

(1)\_\_PRECISION:N 设置精度值，N值范围(1-100)；

(2)\_\_MINRELEVANT:N 设置最小相关度取值，N值范围(1-60)，检索出记录的相关度均要大于N值；

(3)\_\_MINHITWORD:N 必须命中特征词个数，值得注意的事， Wi:+Fi，此种写法权重前有‘+’号，表示必须命中该特征词。

各参数间可以合并使用。需要注意的是，特征词间的分隔符与[参数:N]后的分隔符，必须与系统设置的分隔符保持一致，系统分隔符参数设置参考VECTOR数据类型章节。

检索结果： 返回字段值与“检索词”进行向量运算后，计算结果符合要求的记录。

举例:

1. 前提系统设置(;)为分隔符，查找多个特征词，精度达95%：

SELECT SYS\_VSM FROM CJFD01 WHERE SYS\_VSM ='\_\_PRECISION:95;社会公平问题:7658;和谐社会:6352;实现途径:731;价值理想:544;人类社会:351;社会公正:81;'

1. 前提系统设置(,)为分隔符，查找多个特征词，相关度高于50%的记录：

SELECT SYS\_VSM FROM CJFD01 WHERE SYS\_VSM ='\_\_MINRELEVANT:50,社会公平问题:7658,和谐社会:6352,实现途径:731,价值理想:544,人类社会:351,社会公正:81,'

1. 前提系统设置(;)为分隔符，查找多个特征词，至少命中其中2个特征词，且包含特征词’和谐社会’：

SELECT SYS\_VSM FROM CJFD01 WHERE SYS\_VSM ='\_\_MINHITWORD:2;社会公平问题:7658;和谐社会:+6352;实现途径:731;价值理想:544;人类社会:351;社会公正:81;'

1. 前提系统设置(,)为分隔符，查找多个特征词，至少命中其中2个特征词，且相关度高于50%的记录：

SELECT SYS\_VSM FROM CJFD01 WHERE SYS\_VSM ='\_\_MINHITWORD:2,\_\_MINRELEVANT:50,社会公平问题:7658,和谐社会:6352,实现途径:731,价值理想:544,人类社会:351,社会公正:81,'

### 4.18 FINGERPRINT类型

FINGERPRINT是全文指纹检索类型。

### 4.19 SEARCHENGINE类型

SEARCHENGINE检索类型提供搜索引擎类似的检索功能。

### 4.20 MIX类型

MIX 类型并不提供实际的索引，一般情况下，它给VIRTUAL数据字段实现检索功能。其实现检索的方法是直接查找VIRTUAL数据字段映射的字段的索引。因此，其检索特性由数据字段本身的映射字段的索引来决定。

## 5、数据索引模式

KBASE系统不仅提供了丰富的数据索引类型，来实现各种检索特性，还进一步同时支持多种不同的索引模式，来实现更多的检索特性。

KBASE系统支持的索引模式包括： **NON**（无索引）、**NORMAL**（普通索引）、**UNIQ**（单一索引）、**STEM**（词干索引）。

### 5.1 NON（无索引）

如果是无索引，只能支持精确查询，对正常索引和唯一索引都支持前面提到的所有检索方式。正常索引和唯一索引在使用上是一致的，其差别体现在性能上。唯一索引更适合不重复的字段。

### 5.2 NORMAL（普通索引）(性别)

NORMAL索引模式适合几种数值型索引以外的索引类型。其特征是，在索引结构中每个关键字只记录一次，即关键字唯一。

### 5.3 UNIQ（单一索引）

UNIQ索引模式适合数值型索引和CHAR系列的索引类型。其特征是，在索引结构中每条记录记录一次，这意味着索引结构中关键字可以重复。

### 5.4 STEM（词干索引）

STEM索引模式适合所有的文本型索引类型。其特征是，对英文的数据，会统一按词干进行索引。中文数据会统一繁简来索引，而不同于以前仅仅忽略大小写来索引。

第四章 KSQL数据定义语句

1、数据库定义语句

1.1创建数据库

语法格式：

CREATE DATABASE <数据库名>

参数：

<数据库名>新数据库的名称。数据库名称在服务器中必须唯一，并且符合标识符的规则，数据库名最多可以包含 64个字符。

语句功能：

创建一个新数据库。

使用说明：在KBASE中，数据库是一个逻辑概念，不需独立的存储空间。一个数据库可以包含多个表，但这些表并不需要存储在同一位置。因此，数据库和表之间是一种松散的关系。系统缺省状态下有五个数据库：SYSTEM、DEFAULTDB、USP、TEMP、TELE。

举例：

创建数据库MYDB。

|  |  |
| --- | --- |
| **CREATE DATABASE** | MYDB |
|  |  |

1.2删除数据库

语法格式：

DROP DATABASE <数据库名>

参数：

<数据库名>将要删除的数据库的名称。数据库名称必需在服务器中存在。

语句功能：

删除一个数据库

举例：

删除数据库MYDB。

|  |  |
| --- | --- |
| **DROP DATABASE** | MYDB |

2、表定义语句

2.1创建表

语法格式：

CREATE [<表类型>] TABLE <表定义> [<表的附加定义>] (<列列表>) [WITHSTORAGESPACE<存储空间定义列表>]

<表定义> ::= <表名> [PATH <路径名>] [AT <数据库名>]

<列列表> ::= <列定义> { , <列定义>}

<列定义> ::= <列名><数据存储类型> [(<列长度>)] [<编码>] [INDEX <数据索引类型>] [<索引模式>] [<索引方法>] [ALIASNAME<列别名>] [DISPLAYNAME<列显示名>] [DEFAULT <缺省值>]

<表的附加定义> ::= ALIASNAME <表别名> [<表拥有者><表密码>]

<存储空间定义> ::= <存储空间> {, <存储空间> }

<存储空间> ::= <存储空间类型><存储空间名字> [(<列名> {，<列名>} )]

参数：

<表类型> 不输入表示创建普通表，TEMP 表示创建内存表（临时表）。

<表名> 待创建表的名称，表名在服务器中必需唯一，并且符合标识符的规则，表名最多可以包含 128个字符。

<列名> 列的名称，必需在该表的定义中唯一，并且符合标识符的规则，列名最多可以包含 64个字符。

<路径名> 本表将要存储的物理路径，该路径必需存在，最大长度为256，缺省为KBASE的系统目录，其中路径名的最后一部分是表的物理文件名，路径名中的目录部分一定要存在。

<数据库名> 本表逻辑上隶属的数据库，指定的数据库名必须已经存在。

<列类型> 列的数据类型，有关KBASE数据类型请参照相关章节。

<列长度> 列的存储长度，缺省为4。

<索引模式> 列的索引类型。系统支持的索引模式包括**NON**（无索引）、**NORMAL**（普通索引）、**UNIQ**（单一索引）、**STEM**（词干索引）。缺省为**NON**。

<索引方法> 列的索引方法。系统支持的索引方法有两种：**AUTO**（自动建立索引）和 **MANUAL**（手动索引）

<缺省值> 列的缺省值。

<存储空间类型>**INDEPENDENT** 独立存储空间、**SHARED**共享存储空间。

<存储空间名字> 待创建存储空间的名称。

注：有关存储空间的详细说明见本章的第5节。

语句功能：

创建一个表。

举例：

在C:\DATA目录中创建表STUDENT，包含字段SNO、NAME和AGE，其中在SNO上建有普通索引，该表所属的数据库为MYDB，这里表的物理文件名字与表的名字相同。

|  |  |
| --- | --- |
| **CREATE TABLE** | STUDENT |
| **PATH** | 'C:\DATA\STUDENT' |
| **AT** | MYDB |
|  | (SNO INTEGER NORMAL, NAME CHAR(30),AGE INTEGER) |

2.2修改表结构

#### (1) 添加列

语法格式：

ALTER TABLE <表名> ADD <添加列列表>

<添加列列表>::=<添加单列信息> [,<添加列列表>]

<添加单列信息>::= <列定义> [AT <列号>]

参数：

<列号> 列的位置，缺省在最后。第一列的列号为0，第二列的列号为1，…

<列定义>说明参见创建表的说明。

语句功能：

给指定表添加一列。

举例：

为表STUDENT添加一个ADDRESS列，在该列上建有普通索引，该列位置定为第二列,再添加一个POLITICS列，该列上无索引。

|  |  |
| --- | --- |
| **ALTER TABLE** | STUDENT |
| **ADD** | ADDRESS CHAR(40) NORMAL AT 1, POLITICS CHAR(10) NON |

#### (2) 修改列

语法格式:

ALTER TABLE <表名> ALTER <修改列列表>

<修改列列表>::=<修改单列信息> [,<修改列列表>]

<修改单列信息>::= <原列名> AS <列定义> [AT <列号>]

参数:

参数说明参见创建表、修改列的说明。

语句功能

修改指定表的指定列。

举例:

修改表STUDENT的ADDRESS列，将其列名改为“家庭住址”，不建立索引，别名改为“住址”，位置改为第四列。

|  |  |
| --- | --- |
| **ALTER TABLE** | STUDENT |
| **ALTER** | ADDRESS |
| **AS** | 家庭住址 CHAR(40) NON ALIASNAME 住址 AT 3 |

#### (3) 删除列

语法格式:

ALTER TABLE <表名> DROP <删除列列表>

<删除列列表> ::= <列名>{,<列名>}

参数:

<表名> 待修改的表的名称，表名必须已经存在。

<列名> 列的名称，必须在该表的结构中包含。

语句功能:

删除指定表的指定列。

举例:

删除表STUDENT的“ADDRESS”列和“POLITICS”列。

|  |  |
| --- | --- |
| **ALTER TABLE** | STUDENT |
| **DROP** | ADDRESS, POLITICS |

2.3修改表的物理路径

语法格式：

ALTER TABLE <表名>PATH<新路径名>

参数：

<表名> 待修改的表的名称。

<新路径名>表物理文件的新的路径，将要存储的物理路径，该路径必需存在，最大长度为256，缺省为KBASE的系统目录，其中路径名的最后一部分是表的物理文件名，路径名中的目录部分一定要存在。

语句功能：

修改指定表的物理路径。

使用说明：

表名必须在KBase数据库中已经存在，并且新的存储路径下面存放着完整的物理文件，所谓完整指的是该目录下的信息可以支持KBase以正常的方式处理该表。实际上路径下的物理文件也是KBase系统产生的，只不过是在另一台服务器上产生的，需要做的是将当前KBase系统的表指向新的物理文件。

举例：

表STUDENT的物理路径为D:\KBase\STUDENT，修改其物理路径变为D:\NewData\ STUDENT。

|  |  |
| --- | --- |
| **ALTER TABLE** | STUDENT |
| **PATH** | ‘D:\NewData\STUDENT’ |

2.4删除表

语法格式:

DROP TABLE <表名>

参数:

<表名> 待删除的表的名称，表名必须已经存在。

语句功能:

删除指定表。

举例:

删除表STUDENT。

|  |  |
| --- | --- |
| DROP TABLE | STUDENT |

2.5表的高级操作

#### 2.5.1引入表

语法格式：

IMPORT TABLE <表路径> [ AT <数据库名>]

参数：

<表路径> 表文件的存储路径。

<数据库名> 将表引入后，表所在的数据库名。

语句功能：

引入的表功能起到注册表的作用，前提是存在通过备份或其它手段得到的表的物理文件。如果想重新使用该表，可以通过引入表功能将该表注册到KBase数据库，此时表的实际存储位置仍为对应物理文件所在的位置。

使用说明：

如果指定数据库名，该库名必须存在。在缺省情况下，数据库名为DEFAULTDB。

举例：

引入d:\fts\data下的STUDENT表，引入后该表位于MYDB库。

|  |  |
| --- | --- |
| **IMPORT TABLE** | ‘d:\fts\data\STUDENT’ |
| **AT** | MYDB |

#### 2.5.2连接外部表

语法格式：

CONNECT <表类型> TABLE <表名> PATH <表路径><ip地址><端口号><用户名><口令> [AT <库名>]

<表类型>::= USP | TELE

参数：

<表类型> 只支持连接两种外部表：并行表（TELE）和USP表。

<表名> 欲引入的表的名称。

<表路径> 表的存储路径。

<ip地址> 远程主机的ip地址。

<端口号> 远程主机的端口号。

<用户名> 远程主机的用户名。

<口令> 远程主机的口令。

<库名> 创建后存放的库的名称。

语句功能:

连接指定主机上的外部表，包括并行表（TELE）和USP表等。通过连接外部表可以将其它数据库的内容与KBase数据库发生关联，从而使得用户可以透明的访问其它异构数据源的信息。

使用说明:

必须具有远程主机的用户名和口令才能连接远程机器上的外部表。

举例:

连接192.168.20.52机器上的STUDENT表，连接后的表存储在d:\fts\data下,用户名为guest，口令为guest。

|  |  |
| --- | --- |
| **CONNECT** | TELE |
| **TABLE** | STUDENT |
| **PATH** | 'd:\fts\data' '192.168.20.52' 4567 guest guest |

#### 2.5.3重命名表

语法格式:

DBUM RENAME TABLE <旧表名><新表名>

语句功能:

修改指定表的表名，但是不修改表的文件名。

使用说明:

无。

举例:

修改表STUDENT的表名为STUDENT01。

|  |  |
| --- | --- |
| **DBUM RENAME TABLE** | STUDENT STUDENT01 |

#### 2.5.4修改表的文件名

语法格式：

DBUM RENAME TABLE PATH <表名><目标文件名>

参数：

<目标文件名> 修改表的文件名为指定的目标文件名。

语句功能：

修改指定表的文件名，同时也修改了表名。

使用说明：

无。

举例：

修改STUDENT的文件名为STUDENT01。

|  |  |
| --- | --- |
| **DBUM RENAME TABLE PATH** | STUDENT STUDENT01 |

#### 2.5.5移动表

语法格式:

DBUM RENAME TABLE DATABASE <表名><目标数据库名>

参数:

<目标数据库名> 将表移动到的目标数据库名。

语句功能:

将指定表移动到指定的数据库。

使用说明:

无。

举例:

移动表STUDENT到DEFAULTDB数据库。

|  |  |
| --- | --- |
| **DBUM RENAME TABLE DATABASE** | STUDENT DEFAULTDB |

#### 2.5.6设置表的缓冲级别

语法格式：

DBUM SET TABLE <表名>CACHELEVEL(0|1|2|3|4|5)

参数：

0表示为该表设置系统默认的缓存大小。1表示采用默认大小的2倍，2表示默认大小的4倍，…，5表示默认大小的32倍，最高为5。对于系统表，Hotstar支持自动设置较高的缓存级别，用户可以为经常需要访问且数据量不是太大的表设置较高的缓存级别。

举例：

为STUDENT表设置2级缓冲大小，即是系统默认表缓冲大小的4倍。

|  |  |
| --- | --- |
| **DBUM SET TABLE** | STUDENT |
| **CACHELEVEL** | 2 |

#### 2.5.7激活/禁止表

语法格式：

DBUM ACTIVATE TABLE <表名>(TRUE|FALSE)

参数：

<表名> 将要被激活或禁止的表名。

TRUE表示激活表，FALSE表示禁止表。当一个表被激活时，可以对其进行任何操作。当一个表被禁止时，不能进行查询。在进行大规模索引之前，需要禁止表，以禁止其它用户的访问。再索引建立完成后，需要激活表，才可以被查询。系统关闭时，表当前是否激活的状态并不保存，在下次启动时，每个表的状态都重置为激活。

举例：

禁止表STUDENT。

|  |  |
| --- | --- |
| **DBUM ACTIVATE TABLE** | STUDENT |
|  | FALSE |

2.6创建联合表

语法格式:

CREATE TABLE <联合表定义> AS <子表1> [WITH <子表1> [,<子表2>, …]]

<联合表定义> ::=

<表名> [AT <数据库名>]

参数:

<表名> 待创建表的名称，表名在服务器中必需唯一，并且符合标识符的规则，表名最多可以包含 128个字符。

<数据库名> 本表逻辑上隶属的数据库，指定的数据库名必须已经存在。

说明:

子表1必须存在，子表2，…不一定必须存在，这就提供了一定的扩展，可以以后创建表，创建后该表的信息可通过对应的联合表查找。

子表1、子表2…必须同构，即字段名，字段类型完全一样。

语句功能:

创建一个联合表。

举例:

在数据库MYDB中创建联合表STUDENT，目前已经存在STUDENT01、STUDENT02，并且两个表包含相同的字段SNO、NAME和AGE。

|  |  |
| --- | --- |
| **CREATE TABLE** | STUDENT |
| **AT** | MYDB |
| **AS** | STUDENT01 |
| **WITH** | STUDENT02 |

3、视图定义语句

3.1创建视图

语法格式：

CREATE VIEW <视图名字> AS <SELECT语句>

参数：

<视图名字> 待创建视图的名称，视图名字在服务器中必需唯一，并且由于表和视图共享数据库中相同的名称空间，数据库中不能包含具有相同名称的表和视图，另外，视图名字必须符合标识符的规则，视图名字最多可以包含 128个字符。

<SELECT语句> 参见 KSQL 查询语句的说明。需要注意的值，作为视图的SELECT语句，排序子句、分组子句将被忽略，并且不支持过滤子句。

示例：

创建视图 VSTUDENT。

|  |  |
| --- | --- |
| **CREATE VIEW** | VSTUDENT |
| **AS** | SELECT \* FROM STUDENT WHERE |

3.2删除视图

语法格式：

DROP VIEW <视图名字>

示例：

创建视图 VSTUDENT。

|  |  |
| --- | --- |
| **DROP VIEW** | VSTUDENT |

3.3视图嵌套

KBASE 系统支持两级的视图嵌套。即允许视图作为上面<SELECT语句>中的检索对象，但已经包含视图的视图，不能再被嵌套。

4、索引定义语句

在KBase系统支持四种索引模式：**NON**（无索引）、**NORMAL**（普通索引）、**UNIQ**（单一索引）、**STEM**（词干索引），缺省为**NON**。单一索引指的是在索引树的叶结点中存放的是实际的关键词和对应的单一记录号，普通索引在索引树的叶结点中存放的是指针，指向一个另外的存储单元，存储单元存放着关键词，以及对应该关键词的所有记录号。因此，单一索引适合关键词重复较少的情况，而普通索引适合关键词重复较多的情况。

KBase 系统支持词干索引模式，它主要用于修饰全文类索引。采用此模式，对英文的数据，会统一按词干进行索引，中文数据会统一繁简来索引，而不同于正常情况下仅仅忽略大小写来索引。“词干”索引模式的支持进一步丰富了全文的检索特性。

在KBase系统中，索引的指定是在创建表的时候，并可以指定索引方法。索引方法有两种：**AUTO**（自动建立索引）和 **MANUAL**（手动索引）。这里的KSQL语句提供了单独建立和删除索引的功能，其中的一个原因是全文索引的建立过程通常比较慢，因此对这个过程的操作需要较高的灵活性。

4.1建立索引

语法格式：

INDEX <表名><列名设定>|ALL

[FROM <记录号>]

<列名设定>::=ON <列名> {,<列名>}

参数：

ALL 表示给所有字段建立索引

<列名> 为待建立索引的列。

<记录号> 指定对记录号以后的数据执行索引操作。记录号为整数，从0开始计数，默认为“0”，指从表第一条记录开始索引。[FROM <记录号> ] 可省略，系统将对表或者指定列名的指定开始行以后的数据执行索引。

|  |
| --- |
| **注意：**这里的ALL对应于表在创建时指定索引的所有字段，同样，<列名>也必须在创建表的时候指定了索引。 |

语句功能：

对指定表建立索引。

举例：

给STUDENT表重新建立索引。

|  |  |
| --- | --- |
| **INDEX** | STUDENT all |

语句功能：

对指定表的指定列建立索引。

举例：

给STUDENT表中姓名重新建立索引。索引类型是表定义时已经定义好的。

|  |  |
| --- | --- |
| **INDEX** | STUDENT ON NAME |

语句功能：

对指定表指定列的指定开始行以后的数据建立索引。

举例：

STUDENT表中原有记录500条（已做索引），之后导入了600条记录，给STUDENT表中姓名从500条以后的数据建立索引。索引类型是表定义时已经定义好的。

|  |  |
| --- | --- |
| **INDEX** | STUDENT ON NAME |
| **FROM** | 500 |

4.2删除索引

语法格式:

* + 1. 删除指定表的所有列的索引：

DBUM DROP ALL INDEX OF TABLE <表名>

* + 1. 删除指定表指定列的索引：  
       DBUM DROP INDEX <列名> OF TABLE <表名>

举例:

删除表STUDENT中篇名列的索引。

|  |  |
| --- | --- |
| **DBUM DROP INDEX** | 篇名 |
| **OF TABLE** | STUDENT |

4.3激活/禁止表的索引

语法格式：

DBUM ACTIVATE INDEX <列名> OF TABLE <表名>(TRUE|FALSE)

参数：

TRUE表示激活表指定列的索引，FALSE表示禁止表指定列的索引。表某列索引被禁止后，不能再通过该列查询该表。只有等待该索引被激活，才能根据其查询该表。Hotstar退出时，各表各索引是否激活的信息并不被保存。Hotstar重新启动时，各表各索引自动处于激活状态。

举例：

禁止表STUDENT篇名列的索引。

|  |  |
| --- | --- |
| **DBUM ACTIVATE INDEX** | 篇名 |
| **OF TABLE** | STUDENT |
|  | FALSE |

5、存储空间定义语句

存储空间旨在提高读取数据的速度，其基本原理是将一个表内的数据存储位置按照列的不同分成多个部分，用户检索时可能不需要访问所有部分，以此减少不必要的磁盘I/O。

存储空间的基本操作。

5.1 定义存储空间

#### 5.1.1 在创建表时指定存储空间

在创建表时，可以创建多个存储空间(但有限度，一个表最多六个存储空间)，可以指定每个列存储在那个存储空间上，如不指定，将存储在缺省的存储空间上。

创建表时定义存储语法的语法，见本章2.1节。

#### 5.1.2 创建表后定义存储空间

语法格式：

CREATESTORAGESPACE [SHARED/INDEPENDENT] <存储空间名>ONTABLE<表名>

说明：

SHARED 表示创建的存储空间与默认的存储空间使用共同的.SOB和.LXT文件。

INDEPENDENT 表示创建的存储空间使用独立的.SOB和.LXT文件。

例1：

|  |  |
| --- | --- |
| **CREATE TABLE** | STUDENT (AGE INTEGER NORMAL, SNO INTEGER NORMAL) |
| **WITH STORAGESPACE** | SHARED |
|  | S1(SNO) |

例2：

|  |  |
| --- | --- |
| **CREATE TABLE** | STUDENT01 (AGE INTEGER NORMAL, SNO INTEGER NORMAL) |
| **WITH STORAGESPACE** | INDEPENDENT |
|  | S2(SNO) |

例3：

|  |  |
| --- | --- |
| **CREATE STORAGESPACE** | SHARED |
|  | S3 |
| **ON TABLE** | STUDENT02 |

说明：

存储空间创建后，会在与表的路径下面创建相应的文件。如例1会产生student\_s1.rec文件，例2会产生student01\_s2.rec，student01\_s2.sob，student01\_s2.lxt文件。

#### 5.1.3 添加某列到指定的存储空间

(1) 添加一列

ALTERTABLE<表名> ADD<列定义>[ONSTORAGESPACE <存储空间名>]

(2) 添加多列

ALTERTABLE<表名> ADD<列定义1> [ONSTORAGESPACE <存储空间名1>],…, <列定义N>[ON STORAGESPACE <存储空间名N>]

说明：

指定的表中必须已经创建了该存储空间。

例如：

|  |  |
| --- | --- |
| **ALTER TABLE** | STUDENT |
| **ADD** | COL2 INTEGER |
| **ON STORAGESPACE** | S1 |

说明：

在指定的存储空间添加列后，该列的数据都会存储在该存储空间上。

#### 5.1.4删除存储空间上的列

删除列时不用指定存储空间名。要删除的列必须是存在的列。

1. 删除一列：

ALTER TABLE <表名> DROP <列名>

1. 删除多列：

ALTER TABLE <表名>DROP <列名1>,…,<列名N>

例如：

|  |  |
| --- | --- |
| **ALTER TABLE** | STUDENT |
| **DROP** | COL1 |

5.2 删除存储空间

语法格式：

DROPSTORAGESPACE <存储空间名> on table <表名>

说明：

存储空间上必须没有任何列，才能进行DROP操作。DROP后，相应的文件还存在，但是该文件与该表已经没有任何关系。

例如：

|  |  |
| --- | --- |
| **DROP STORAGESPACE** | S1 |
| **ON TABLE** | STUDENT |

6、索引空间定义语句

6.1 创建一个索引空间

语法格式：

CREATE INDEXSPACE <索引空间名>

ON TABLE <表名>

6.2 修改一个或多个索引的索引空间位置

语法格式：

DBUM MOVE INTO INDEXSPACE <索引空间名>

<索引列名>[,<索引列名>… ]

ON TABLE <表名>

例如：

|  |  |
| --- | --- |
| **CREATE INDEXSPACE** | 姓名 |
| **ON TABLE** | STUDENT |

|  |  |
| --- | --- |
| **DBUM MOVE INTO INDEXSPACE** | 姓名 |
|  | NAME |
| **ON TABLE** | STUDENT |

|  |  |
| --- | --- |
| **CREATE INDEXSPACE** | 频次 |
| **ON TABLE** | 期刊 |

|  |  |
| --- | --- |
| **DBUM MOVE INTO INDEXSPACE** | 频次 |
|  | 被引频次,他引频次,下载频次,浏览频次,是否高下载,是否高浏览,是否高被引,是否高他引 |
| **ON TABLE** | 期刊 |

7、字段映射关系定义语句

7.1 添加字段映射关系

语法格式：

ALTER TABLE <表名>ADD FIELDMAP

<映射字段名><源字段名><权重>

[, <映射字段名><源字段名><权重> ]

[, <映射字段名><源字段名><权重> ]

举例：

添加表STUDENT的字段映射关系：源字段NAME映射到字段SNO，权重为1

|  |  |
| --- | --- |
| **ALTER TABLE** | STUDENT |
| **ADD FIELDMAP** | SNO |
|  | NAME |
|  | 1 |

7.2删除字段映射关系

语法格式：

ALTER TABLE <表名>DROP FIELDMAP

<映射字段名><源字段名>

[, <映射字段名><源字段名>]

[, <映射字段名><源字段名>]

举例：

删除表STUDENT中NAME和SNO的字段映射关系

|  |  |
| --- | --- |
| **ALTER TABLE** | STUDENT |
| **DROP FIELDMAP** | SNO |
|  | NAME |

第五章 KSQL数据查询语句

KBase功能特性，主要通过KBase提供的数据查询功能来体现。如同标准SQL语法，KSQL的查询语法，和 SQL的查询语法类似，下面详细阐述。

1、单表查询

语法格式：

SELECT <选择列表>

FROM <表名>

[<WHERE 子句>]

[<FILTER BY 子句>]

[<FILTER BY NESTEDFILTER 子句>]

[<GROUP BY 子句>]

[<ORDER BY 子句>]

其中：

<WHERE 子句>::= WHERE <条件项> { AND|OR|NOT <条件项>}

<表名> ::= <表名> | <视图名>

<条件项> ::= (<条件项> { AND|OR|NOT <条件项>} )

<条件项> ::= [<表名>.]列名 <比较符><检索关键词>

<条件项> ::= [<表名>.]列名 = | % <复合检索关键词>

<条件项> ::= [<表名>.]列名 = XLS(<检索关键词>[,扩展词典])

<条件项> ::= [<表名>.]列名 IS NULL

<比较符> ::= > | < | >= | <=

<复合检索关键词> ::= <单元检索关键词> {+|-|\* <单元检索关键词>}

<单元检索关键词> ::= <检索关键词> | (<复合检索关键词>)

参数：

<选择列表> 需要返回的投影列，可以用’\*’表示需要返回所有列。当前系统，在指明投影列时，最多可以返回 **1024** 个列。

<表名> 待查询的表的名称或视图名称，相应的表或视图必须已经存在。

<视图名> 待查询的视图的名称，视图名必须已经存在。

<检索关键词> 检索条件中的检索关键词，应该与列的类型、其支持的检索语法一致，如对整数型列的条件不应该为字符串。不同的索引类型所支持的检索语法，参见各种索引数据类型的说明。

<复合检索关键词> 是一种复杂、高效的检索表示，+ 表示取或(OR),- 表示取非(NOT),\*表示取(AND)。

XLS(<检索关键词>[,扩展词典]) 是检索词自动扩展查询，系统利用内部词典(或用户指定的词典)，扩展用户的检索词，根据内部扩展词典的配置，可以自动进行中英文扩检、同义词扩检等。

<FILTER BY 子句> 详细说明见后面“过滤查询”章节。

<FILTER BY NESTEDFILTER子句> 详细说明见后面“嵌套过滤查询”章节。

<GROUP BY 子句> 详细说明见后面“分组查询”章节。

<ORDER BY 子句> 详细说明见后面“排序查询”章节。

语句功能：

检索指定的表或视图。

举例：

设表“CJFD2004”存储了2004年中国期刊文章，其创建表的语句为：

|  |  |
| --- | --- |
| **CREATE** |  |
| **TABLE** | CJFD2004 |
|  | (  篇名 VCHAR(255) ASCII INDEX TITLE NORMAL,  作者 MVCHAR(32767) ASCII INDEX MSTRCHAR NORMAL,  机构 MVCHAR(32767) ASCII INDEX MSTRCHAR NORMAL,  关键词 MVCHAR(32767) ASCII INDEX MSTRCHAR NORMAL,  中文摘要MTEXT(32767) ASCII INDEX ABSTRACT NORMAL,  全文 LTEXT ASCII INDEX LTEXT NORMAL),  引文 LTEXT(2147483647) ASCII INDEX COMPACTQTXT NORMAL,  被引频次 INTEGER(8) ASCII INDEX INTEGER UNIQ,  分类号 MVCHAR(32767) ASCII INDEX MVCHAR NORMAL ALIASNAME 中图分类号,  专题子栏目代码 MVCHAR(32767) ASCII INDEX EMVCHAR NORMAL,  中文刊名 MVCHAR(32767) ASCII INDEX MSTRCHAR NORMAL ALIASNAME 显示来源,  出版日期 DATE(16) ASCII INDEX QDATE UNIQ ALIASNAME 发表时间,  文件名 CHAR(24) ASCII INDEX CHAR UNIQ  ) |

下面的检索都是针对“CJFD2004”表进行。

检索一： 检索全部文章。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |

检索二： 检索篇名出现了“计算机”的文章。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名=”计算机” |

检索三： 检索篇名出现了“计算机”或“电脑”的文章。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名=”计算机”+”电脑” |

检索四： 检索篇名出现了“计算机”并且是“清华大学”发表的文章。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名=”计算机” AND 机构=”清华大学” |

2、多表查询

语法格式：

SELECT <选择列表>

FROM <表名列表>

[<WHERE 子句>]

[<FILTER BY 子句>]

[<FILTER BY NESTEDFILTER 子句>]

[<GROUP BY 子句>]

[<ORDER BY 子句>]

其中：

<表名列表>::= <表名> {,<表名>}

<表名> ::= <表名> | <视图名>

参数：

<表名列表> 中的多个表或视图应该具有相同的结构，如果结构不一致，最好在<选择列表>中指明需要返回的列，如果用‘\*’作为<选择列表>，KBASE 系统会默认<表名列表>所有的表与第一个表的结构相同，并以第一个表的列作为返回结果集的列。

其它参数的说明请参见“单表查询”一节。

语句功能：

同时检索多个表或视图。

举例：

设表“CJFD2004”存储了2004年中国期刊文章，“CJFD2003”存储了2003年中国期刊文章。检索篇名出现了“计算机”并且是清华大学发表的2003和2004两年的文章，KSQL查询语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004,CJFD2003 |
| **WHERE** | 篇名=”计算机” AND 机构=”清华大学” |

3、跨库查询

跨库查询是指在一次检索中，同时指定多个库作为检索目标，其查询结果等价于分别查询各库得到查询结果的“并集”。所谓“并集”，是指跨库查询不是简单的将多个库的查询结果混合到一起，它对查询结果做了“排重”处理。排重的任务是识别多个库间的“重复记录”，只保留一条记录在结果集中。

跨库查询时，多个库有主库和从库之分。当所有主库的查询都完成时，跨库查询才可以返回，但跨库查询返回时并不要求从库的查询都执行完毕。主库和从库往往有两个区别：

1. 主库的优先级别高于从库的优先级别。用户虽然递交的是跨库查询语句，但是，在查询结果中，他首先期望看到的结果是主库中的结果，其次才是从库中的结果。当结果分页显示时，主库的结果占去整页的大部分条目，从库的结果只有一小部分条目。
2. 主库查询的速度高于从库的速度。一般指定本地库、并行库为主库，而USP库为从库。比如用户同时检索“中国学术期刊全文数据库”和“重庆维普期刊数据库”。前者是本地库，因而应为主库，后者是USP库，因而应为从库。

进行跨库查询时，还需要指明哪些库之间是有重复记录的。Hotstar数据库引擎认为主库之间是没有重复记录的，所有从库可能和某几个主库有重复记录，至于是哪些主库，用户需要指明。

排重字段是判定两条记录是否重复的依据，Hotstar数据库引擎依据两条记录中该字段的值是否相等，来判定这两条记录是否重复。

KBase数据库针对跨库查询提供了专门的KSQL语句：

语法格式：

VARSELECT <选择列表>

FROM <库名列表> | <视图名>

[<WHERE 子句>]

[<FILTER BY 子句>]

[<FILTER BY NESTEDFILTER 子句>]

[<GROUP BY 子句>]

[<ORDER BY 子句>]

WITH <主库信息> ',' <重复库信息> ',' <排重字段信息>

其中:

<WHERE 子句>::= WHERE <条件项> { AND|OR|NOT <条件项>}

<条件项> ::= (<条件项> { AND|OR|NOT <条件项>} )

<条件项> ::= [<表名>.]列名 <比较符><检索关键词>

<条件项> ::= [<表名>.]列名 =|% <复合列值>

<条件项> ::= [<表名>.]列名 = XLS(<检索关键词>[,扩展词典])

<比较符> ::= >|<|>=|<=

<复合列值> ::= <单元列值> {+|-|\* <单元列值>}

<单元列值> ::= <检索关键词> | (<复合列值>)

<库名列表> ::= <库名> {<库名>}

<主库信息> ::= MAINDB ‘=’ (<主库列表>|NULL)

<主库列表> ::= ‘(‘<库名> {<库名>} ‘)’ |<库名>

<重复库信息> ::= DUPDB ‘=’ (<重复库列表>|NULL)

<重复库列表> ::= ‘(‘<库名> {<库名>} ‘)’ |<库名>

<排重字段信息> ::= REFCOL ‘=’ ( <列名>| NULL)

参数：

<库名> 可以是视图名，也可以是表名。

<主库信息> 指定哪些是主库，可以为NULL，表示没有主库。

<重复库信息> 指定哪些主库与从库重复，可以为NULL，表示所有主库都不与从库重复。

<排重字段信息> 指定排重字段，可以为空，表示不排重。

举例：

查询一

查询“中国学术期刊全文数据库”和“重庆维普期刊数据库”中篇名包含“空间数据库”的所有文章，主库是“中国学术期刊全文数据库”，“中国学术期刊全文数据库”和“重庆维普期刊数据库”有重复记录，需要排重，排重字段是篇名。

语句举例：

|  |  |
| --- | --- |
| **VARSELECT** | \* |
| **FROM** | 中国学术期刊全文数据库, 重庆维普期刊数据库 |
| **WHERE** | 篇名=”空间数据库” |
| **WITH** | MAINDB=中国学术期刊全文数据库, DUPDB=中国学术期刊全文数据库, REFCOL=篇名 |

查询二

查询“中国学术期刊全文数据库”， “中国优秀博硕士学位论文全文数据库”和“重庆维普期刊数据库”中关键词包含“空间数据库”的所有文章，主库是“中国学术期刊全文数据库”和“中国优秀博硕士学位论文全文数据库”，“中国学术期刊全文数据库”和“重庆维普期刊数据库”有重复记录，需要排重，排重字段是篇名。”

|  |  |
| --- | --- |
| **VARSELECT** | \* |
| **FROM** | 中国学术期刊全文数据库, 中国优秀博硕士学位论文全文数据库，重庆维普期刊数据库 |
| **WHERE** | 关键词=”空间数据库” |
| **WITH** | MAINDB=(中国学术期刊全文数据库, 中国优秀博硕士学位论文全文数据库), DUPDB=中国学术期刊全文数据库, REFCOL=篇名 |

查询三

查询“重庆维普期刊数据库”和“GOOGLE”中关键词包含“空间数据库”的所有文章，没有主库，因而不需要排重。”

|  |  |
| --- | --- |
| **VARSELECT** | \* |
| **FROM** | 重庆维普期刊数据库，GOOGLE |
| **WHERE** | 关键词=”空间数据库” |
| **WITH** | MAINDB=NULL, DUPDB=NULL, REFCOL=NULL |

4、排序查询

语法格式：

<ORDER BY 子句>::= ORDER BY [TOP N ]<排序列>

参数：

<排序列>::= RELEVANT

<排序列>::= <DEC|DATE>

<排序列>::= <列名> [ASC|DESC]

<排序列>::= (<列名> , <排序词典名>）[ASC|DESC]

<排序列> 为RELEVANT表示按照相关度排序。

<排序列> 为DEC或DATE表示按照数据添加顺序倒排。

语句功能：

单独使用排序<ORDER BY 子句>时，可按照一列或两列排序；如果和分组<GROUP BY 子句>并列使用，只能按照一列排序。如使用聚函数或count(\*)排序 ，那聚函数或count(\*)必须存在于SELECT <选择列表>中。

举例：

查询：查找篇名含有“计算机”的文章，并按照相关度排序。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名=计算机 |
| **ORDER BY** | RELEVANT |

TOP N:

KBase 作为管理海量信息的数据库系统，其查询结果集往往会比较大，为提高性能和可用性，系统排序默认返回前2000条结果。也就是说，在默认情况下，前2000条结构排序是有序的，2000条以后的结果不保证有序。

例如：

|  |  |
| --- | --- |
| **SELECT** | 被引频次 |
| **FROM** | CJFD2004 |
| **ORDER BY** | 被引频次 Desc |

返回的结果是，其前面 2000 条从大到小排列，2000条以后结果顺序不保证，但2000条以后的结果，“被引频次”值都是小于或等于第2000条记录的“被引频次”值。

如果用户要求结果中，有序的范围更大，可以使用TOP N 条件。如上例，如果希望前3000条有序，则可以写成：

|  |  |
| --- | --- |
| **SELECT** | 被引频次 |
| **FROM** | CJFD2004 |
| **ORDER BY** | TOP 3000 被引频次 Desc |

如果希望修改默认情况下的排序个数，可以修改系统参数 ORDERTOPN。修改方法参照第七章 5.1 节。

5、分组查询

语法格式：

<GROUP BY 子句>::= <分组列>

参数：

<分组列>::= <列名> [ASC|DESC]

<分组列>::= (<列名> , <排序词典名>）[ASC|DESC]

<分组列>::=(<列名> , <排序词典名>,CROSS）

<分组列> ASC表示升序，DESC表示降序，缺省为未定义。

<分组列> CROSS表示交叉分组。

语句功能：

按照指定列分组，且只支持一列分组。KBASE中的分组与常规关系数据库基本相同。

交叉分组查询适用于多值字段。具体实现为对多值字段的子值进行两两组合，统计同时包含两个子值的分组结果，且分组结果默认按照COUNT(\*)降序排列。交叉分组查询结果成对出现，即相邻两行显示一个交叉分组，且这两行分组统计COUNT(\*)结果相同。另外，使用交叉分组后不支持排序。

举例：

查询：按照机构对CJFD2004分组。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **GROUP BY** | 机构 |

查询：按照专题子栏目代码对CJFD2000交叉分组。

|  |  |
| --- | --- |
| **SELECT** | 专题子栏目代码，COUNT(\*) |
| **FROM** | CJFD2000 |
| **GROUP BY** | (专题子栏目代码，CCL3，CROSS) |

6、过滤查询

语法格式：

< FILTER BY 子句> ::= FILTER BY (<列名>, <查询关键字>)

参数：

<查询关键字> ::= “<关键字>[,<关键字>[…]]”

说明：

其中，“查询关键字”可以为一个或多个关键词，或多个关键词用半角逗号分隔，多个关键字作为一个语法记号，用单引号或双引号引起。

过滤查询是KBASE系统提供的一种查询优化的方法。

举例1：

|  |  |
| --- | --- |
| **SELECT** | 篇名,分类号 |
| **FROM** | CJFD2004 |
| **WHERE** | 分类号 = "A?" OR分类号 = "B?" |

利用过滤查询可以写作：

|  |  |
| --- | --- |
| **SELECT** | 篇名,分类号 |
| **FROM** | CJFD2004 |
| **FILTER BY** | (分类号, "A?, B?" ) |

在语法上需要注意的是，FILTER BY 子句是 WHERE 子句的补充，它总是记在WHERE 子句的后面，无论是在单库检索语句中还是在跨库检索语句中都是如此。

举例2：

在CNKI总库平台、kns等系统中，检索页面为用户提供了“学科导航”的控制方法。用户可以选择多个学科做为控制条件来约束检索结果。在传统的实现方案中，页面通过如下的ksql形式发送检索请求:

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 全文=计算机 and (专题子栏目代码= A?+B01?+C?+E?+F?) |

其中“专题子栏目代码= A?+B01?+C?+E?+F?”表示用户选择了多个类别。这种实现方式有着极大的弊端，每个子条件(A?,B01等)都将产生一个很大的中间结果，对这些中间结果进行AND运算将耗费很多的CPU资源，使整个系统的响应能力很差。

采用过滤查询方案，将能极大提高性能：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 全文=计算机 |
| **FILTER BY** | (专题子栏目代码, ‘A?,B01?,C?,E?,F?’ ) |

7、嵌套过滤查询

语法格式：

< FILTER BYNESTEDFILTER 子句> ::=

FILTER BYNESTEDFILTER (<列名1> [NOT] IN <NESTED SELECT 子句>)

<NESTED SELECT 子句> ::=

SELECT <列名2> FROM <表名> [<WHERE 子句>]

参数：

<列名1> 是主检索语句的有效列名。

<列名2> 是NESTED SELECT 子句的有效列名。

<WHERE 子句> 同前面单表检索。

说明：

嵌套过滤查询是KBASE系统提供的另一种过滤查询方法。

其中<列名 1>、<列名 2>对存储类型有限制，支持的存储类型包括INTEGER、INT64、DATE、NUM、TIME、AUTO、CHAR、VCHAR、MVCHAR、LMVCHAR。

内层查询的SELECT查询总使用圆括号括起来。

外层查询支持多表及视图 ，内层查询只能是单表，且均不支持远程表。

子查询中一般不使用<group by>、<order by>子句，只能对最终查询结果进行分组排序。

举例：

这里期刊库和作者库的结构如下：

|  |  |
| --- | --- |
| **CREATE** |  |
| **TABLE** | CJFD2004 |
|  | (  篇名 VCHAR(255) ASCII INDEX TITLE NORMAL,  作者 MVCHAR(32767) ASCII INDEX MSTRCHAR NORMAL,  作者代码 CHAR(20) ASCII INDEX CHAR NORMAL,  机构 MVCHAR(32767) ASCII INDEX MSTRCHAR NORMAL,  关键词 MVCHAR(32767) ASCII INDEX MSTRCHAR NORMAL,  中文摘要MTEXT(32767) ASCII INDEX ABSTRACT NORMAL,  全文 LTEXT ASCII INDEX LTEXT NORMAL),  引文 LTEXT(2147483647) ASCII INDEX COMPACTQTXT NORMAL,  被引频次 INTEGER(8) ASCII INDEX INTEGER UNIQ,  分类号 MVCHAR(32767) ASCII INDEX MVCHAR NORMAL ALIASNAME 中图分类号,  专题子栏目代码 MVCHAR(32767) ASCII INDEX EMVCHAR NORMAL  ) |

|  |  |
| --- | --- |
| **CREATE** |  |
| **TABLE** | CNKI\_SCHOLAR |
|  | (  作者 MVCHAR(32767) ASCII INDEX MSTRCHAR NORMAL,  作者代码 CHAR(20) ASCII INDEX CHAR NORMAL,  作者机构代码 CHAR(20) ASCII INDEX CHAR NORMAL,  学者单位 MVCHAR(32767) ASCII INDEX MSTRCHAR NORMAL,  学者职称 MVCHAR(32767) ASCII INDEX MVCHAR NORMAL,  文献篇数 INTEGER(8) ASCII INDEX INTEGER UNIQ,  职务 MVCHAR(32767) ASCII INDEX MVCHAR NORMAL,  职务级别 MVCHAR(32767) ASCII INDEX MVCHAR NORMAL,  科技成果文献数 INTEGER(8) ASCII INDEX INTEGER UNIQ)  ) |

从“CJFD2004”中检索“篇名”中含有“激光”，并且其作者所属机构是“清华大学”职称为“教授”发表文献大于200篇作者的文献。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名=激光 |
| **FILTER BY** | NESTEDFILTER ( 作者代码 in  select 作者代码 FROM CNKI\_SCHOLAR WHERE 学者单位='清华大学'  AND 学者职称=教授  AND 文献篇数>200 ) |

从“CJFD2004”中检索“篇名”中含有“激光”，并且其作者职称非“教授”发表文献小于200篇作者的文献。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名=激光 |
| **FILTER BY** | NESTEDFILTER ( 作者代码 not in  select 作者代码 FROM CNKI\_SCHOLAR WHERE 学者职称=教授  AND 文献篇数>=200 ) |

8、其他 KSQL高级查询

8.1聚集函数查询

语法格式：

SELECT <聚集函数>

FROM <表名列表>

[<WHERE 子句>]

[<GROUP BY 子句>]

[<ORDER BY 子句>]

参数：

系统支持的聚集函数如下表，参与聚集运算的列必须是数值型列。

|  |  |
| --- | --- |
| **函数** | **含义** |
| MIN(列名) | 对结果集中指定的列，求最小值 |
| MAX(列名) | 对结果集中指定的列，求最大值 |
| SUM(列名) | 对结果集中指定的列，求和 |
| AVG(列名) | 对结果集中指定的列，求均值 |

语句功能：

对一列求聚集函数。

举例：

求学生表中的所有学生的平均年龄。

|  |  |
| --- | --- |
| **SELECT** | AVG(AGE) |
| **FROM** | STUDENT |

8.2复杂组合查询

复杂组合查询是采用复合列的一种高级查询方式，用以帮助用户进行复杂检索功能的实现。<复合检索关键词> 是一种复杂、高效的检索表示，是将列通过运算符组合的表示形式。共有三种运算符：+、-和\*，其中+ 表示或(OR),- 表示非(NOT),\*表示与(AND)。

举例：

检索篇名出现了“计算机”但没有出现“数据库”的文章，KSQL检索语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名=‘计算机’-‘数据库’ |

检索全文中出现了“计算机科学”或“信息科学”，但是没有出现“人文科学”，且作者姓刘的文章。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 全文=计算机科学+信息科学-人文科学 AND 作者%刘 |

检索篇名同时出现了“计算机”和“软件”的文章，KSQL检索语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名= 计算机\*软件 |

8.3自动扩展查询

XLS(<检索关键词>[,扩展词典]) 是自动扩展查询，如 检索关键词中出现了“非典”或其同义词，并且是北京大学医学院发表的文章。

举例：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 关键词=XLS(非典) and 机构=北京大学医学院 |

如果系统默认自动扩展词典包含词条对：“非典”、“SARS”，则上述检索相当于

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | (关键词=非典 or 关键词=SARS) and 机构=北京大学医学院 |

扩展词典是系统中一张有特定格式的表，系统默认的扩展词典是表 SYS\_XLDICT。该表的格式要求是，其至少包含两列，系统自动在首列上查询带扩展的关键字，第二列提供扩展词。这两列的列名系统没有特定的要求。

假定用户有自定义的扩展词典USER\_DICT，则可以用如下语句使用这个词典：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 关键词=XLS(非典, USER\_DICT) and 机构=北京大学医学院 |

8.4模糊查询

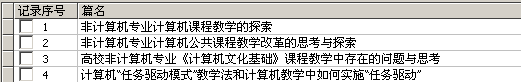
%是模糊查询符，模糊查询并不考虑检索条件中词的出现顺序，如检索期刊中出现“计算机”和“教学”的文章，下面的两个KSQL语句是等价的。

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名 % “计算机教学” |

和

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | CJFD2004 |
| **WHERE** | 篇名 % “教学计算机” |

对应的查询结果如下：



8.5 BELONG TO查询

KBASE系统在检索语句的WHERE 子句中的条件项中支持 BELONG TO 查询。

语法如下：

<条件项> ::= [<表名>.]列名 BELONG\_TO （<查询关键字>）

其中，“查询关键字”可以为一个或多个关键词，或多个关键词用半角逗号分隔。形如： “中国, 美国,俄罗斯”

举例：

|  |  |
| --- | --- |
| **SELECT** | 篇名,分类号 |
| **FROM** | CJFD |
| **WHERE** | 分类号 = "A?" OR分类号 = "B?" |

利用BELONG TO 可以写作：

|  |  |
| --- | --- |
| **SELECT** | 篇名,分类号 |
| **FROM** | CJFD |
| **WHERE** | 分类号 BELONG\_TO ( "A?, B?" ) |

8.6 函数查询

KBASE系统在检索语句的SELECT 子句中，还支持其他特定的查询函数，以实现特定的功能。

#### 8.6.1 函数GETSYSFIELD、GETSNAPSHOT

用法含义同第三章2.11节所述。

|  |  |  |
| --- | --- | --- |
| 类型 | 意义 | 替换函数 |
| TNAME | 返回表名称 | GETSYSFIELD(‘\_ \_TABLENAME ’ ) |
| TANAME | 返回表的显示名称 | GETSYSFIELD(‘\_ \_TABLEALIASNAME’) |
| RECORDID | 返回物理记录号 | GETSYSFIELD(‘\_ \_RID’) |
| RELEVANT | 返回检索结果的相关度 | GETSYSFIELD(‘\_ \_RELEVANT’) |
| SNAPSHOT | 返回与检索相关的特定字段的检索快照 | GETSNAPSHOT ( ‘被映射的列名’ ) |

#### 8.6.2 函数CODENAME、GROUPCODENAME

CODENAME用法：

CODENAME ( FILENAME[, 转换表[, hFindCol [, hGetCol]]] )

说明：

1. FIELDNAME： 为待转换的字段名，如果为 \_\_GROUPNAME, 则表示将分组组名进行转换；如果为 \_\_GROUPVALUE, 则表示将分组值进行转换。
2. 转换表： 为进行显示转换的表，他包含查找列和显示列.
3. hFindCol： 查找列的列序号
4. hGetCol： 显示列的列序号

举例1：

构造两张表，第一张含有基金代码列V和基金代码对应的分组值I，表名是“test”，另一张是中间转换表SYS\_FUNDCODENAME，含有基金代码列CODE和基金名称NAME，如果检索基金代码分组值对应的基金名称的记录，则其KSQL查询语句如下：

|  |  |
| --- | --- |
| **SELECT** | I,CODENAME(V,SYS\_FUNDCODENAME,0,1) as 基金 |
| **FROM** | test |

举例2：

检索有基金参与的主题及其对应的基金名称的记录，KSQL查询语句如下：

|  |  |
| --- | --- |
| **SELECT** | 主题,基金代码,CODENAME(\_\_GROUPNAME,SYS\_FUNDCODENAME,0,1) as 基金 |
| **FROM** | CJFD2013 |
| **GROUP BY** | (基金代码,minteger) |

GROUPCODENAME是CODENAME的一种简写，用法如下：

用法一： GROUPCODENAME ( )，同 CODENAME ( \_\_GROUPNAME, SYS\_CODENAME, 0, 1 )；

举例：

以下KSQL查询语句的含义一致，表示按照专题子栏目代码排序列分组，取对应的题名及其发文量。第一种是第二种的简写。

|  |  |
| --- | --- |
| **SELECT** | 专题子栏目代码 as 代码,groupcodename() as 题名,count(\*) as 发文量 |
| **FROM** | CJFD2013 |
| **GROUP BY** | (专题子栏目代码,CCL2) |

|  |  |
| --- | --- |
| **SELECT** | 专题子栏目代码 as 代码,codename(\_\_GROUPNAME,SYS\_CODENAME,0,1) as 题名,count(\*) as 发文量 |
| **FROM** | CJFD2013 |
| **GROUP BY** | (专题子栏目代码,CCL2) |

用法二：GROUPCODENAME ( 转换表)，同 CODENAME ( \_\_GROUPNAME, 转换表, 0 , 1 )

举例：

以下KSQL查询语句的含义一致，表示根据机构代码排序列分组，检索与机构代码是"0073968" "0038515" "0859570"的机构有合作的机构及其机构代码，并统计各合作机构的发文量。第一种是第二种的简写。

|  |  |
| --- | --- |
| **SELECT** | 机构代码 as 代码,groupcodename (sys\_instcodename) as 题名,  count (\*) as 发文量 |
| **FROM** | CJFD |
| **WHERE** | 机构代码="0073968"+"0038515"+"0859570" |
| **GROUP BY** | (机构代码,minteger) |
| **ORDER BY** | count (\*) desc |

|  |  |
| --- | --- |
| **SELECT** | 机构代码 as 代码,CODENAME( \_\_GROUPNAME,sys\_instcodename,0,1) as 题名,count (\*) as 发文量 |
| **FROM** | CJFD |
| **WHERE** | 机构代码="0073968"+"0038515"+"0859570" |
| **GROUP BY** | (机构代码,minteger) |
| **ORDER BY** | count (\*) desc |

#### 8.6.3 函数KEYSENTENCE

KEYSENTENCE() 的作用是为了返回关键句子，返回一个XML格式的文本内容，包含多个与检索词相关的句子，并按相关度排序。

用法格式为：KEYSENTENCE (字段名[，N])

字段名一般为“全文”，N表示要求返回的句子数目，如省略默认为10。

用户的检索检索词，系统能自行从检索条件中分析，如果用户没有输入对应的检索词，则此函数返回空的内容。

返回的文本示例为:

**<?xml version = "1.0" encoding="gbk"?>**

**<Text Total="2">-- Total 表示总共返回了多少个句子**

**<S R="325">-- R 表示这个句子的相关度**

**<![CDATA[而利用数据库的方法稍复杂些,但借用数据库技术,能进行多种数据的复杂处理。]]>**

**</S>**

**<S R="212">**

**<![CDATA[而数据库是当今数据管理技术的主要形式和方法,它具有强大的数据处理功能和灵活性。]]>**

**</S>**

**</Text>**

#### 8.6.4 函数GETFIELDLEN

GETFIELDLEN() 的作用是为了返回字段的长度。当数据存储类型为定长时，GETFIELDLEN()返回该存储类型设定长度；当数据存储类型为变长时，GETFIELDLEN()返回数据实际长度。

用法格式为：GETFIELDLEN(字段名)

8.7 分页查询

KBase能自动处理超大的结果集，如果用户希望将结果集按页进行切分，KBase通过两个SQL指令来实现。需要注意的是，KBase 每页最大记录数是 512，如果结果集超过512，则最多只返回512条记录。

#### 8.7.1 TOP 选项

如同标准SQL，用户可以在SELECT 后书写 TOP N， 返回前面 N 条记录。当N=0时，返回全部记录。

例如：

|  |  |
| --- | --- |
| **SELECT** | TOP 10 score |
| **FROM** | student |
| **ORDER BY** | score |

该语句返回分数最高的10名同学的分数。

|  |  |
| --- | --- |
| **SELECT** | TOP 0 \* |
| **FROM** | student |

该语句返回全部记录，且最多返回512条记录。即当student记录数超过512时，只返回512条记录。

#### 8.7.2 LIMIT选项

TOP N 相当于返回第一页的数据，如果用户想返回其他页的数据，可以在SQL语句最后使用LIMIT F, N 选型,其中 F 表示从第几条记录开始（注意，F从0开始），N表示返回多少条记录。当N=0时，返回从第F条记录开始的所有记录。

例如：

|  |  |
| --- | --- |
| **SELECT** | score |
| **FROM** | student |
| **ORDER BY** | score |
| **LIMIT** | 20, 10 |

该语句返回同学分数的第21名到30名。

|  |  |
| --- | --- |
| **SELECT** | score |
| **FROM** | student |
| **ORDER BY** | score |
| **LIMIT** | 20, 0 |

该语句返回同学分数的第21名到最后的记录，且最多返回512条记录。即当第21名到最后总记录数超过512时，只返回512条记录。

LIMIT 还有一种用法是 LIMIT N，其含义同TOP N。

TOP与LIMIT同时使用规则：

1、TOP N与LIMIT M同时存在，等价于取LIMIT M结果集再做TOP N操作。

2、TOP N与LIMIT F,M同时存在，等价于取LIMIT F,M结果集再做TOP N操作。

例如：

|  |  |
| --- | --- |
| **SELECT** | TOP 10 score |
| **FROM** | student |
| **LIMIT** | 20 |

该语句返回同学分数的前10条记录。

|  |  |
| --- | --- |
| **SELECT** | TOP 10 score |
| **FROM** | student |
| **LIMIT** | 20, 100 |

该语句返回同学分数的第21到30条记录。

第六章 KSQL数据操作语句

1、插入记录

语法格式：

INSERT INTO <表名> (

<列名> [,<列名>] )

VALUES (

<检索关键词> [,<检索关键词>])

参数：

<列名> 必须与列值的数目一致，并且一一对应。

语句功能：

向指定表插入一条记录。

举例：

向STUDENT表插入姓名为张三，年龄为15的记录。

|  |  |
| --- | --- |
| **INSERT INTO** | STUDENT |
|  | (NAME,AGE) |
| **VALUES** | (张三,15) |

2、更新记录

语法格式：

UPDATE <表名>

SET <列值设定>

[<WHERE 子句>]

参数：

<列值设定> ::=<单列列值设定> {,<单列列值设定>}

<单列列值设定> ::= <列名> = <检索关键词>

<检索关键词>::= <检索词>|<复合检索关键词>

<复合检索关键词> ::= <表达式函数运算>

<表达式函数运算> ::= <表达式函数+表达式函数>|

<表达式函数-表达式函数>|

<表达式函数\*表达式函数>|

<表达式函数/表达式函数>|

<-表达式函数>

<WHERE 子句>语法同SELECT 语句。

<单列列值设定>中，<列名>必须与<检索关键词>类型兼容。表达式函数运算中，加、减、乘、除、取反适用于数值（存储类型：INTEGER、NUM、INT64）运算，且加也适用于字符运算（表示连接字符串）。

系统支持的表达式函数如下表：

|  |  |
| --- | --- |
| **函数** | **含义** |
| GETDATE() | 返回当前日期 |
| DATEADD(interval, number, date) | 返回指定时间间隔的日期 |
| UPPER(字段名) | 将字符串转换成大写 |
| LOWER(字段名) | 将字符串转换成小写 |
| LEFT(字段名, 长度) | 将字符串的左边指定长度的字节截取返回 |
| RIGHT(字段名, 长度) | 将字符串的右边指定长度的字节截取返回 |
| LTRIM(字段名) | 去除字符串前面的空格 |
| LTRIM(字段名, 字符串) | 去除字符串前面的指定字符 |
| RTRIM(字段名) | 去除字符串后面的空格 |
| RTRIM(字段名, 字符串) | 去除字符串后面的指定字符 |
| TRIM(字段名) | 去除字符串前后空格 |
| TRIM(字段名, 字符串) | 去除字符串前后指定字符 |
| SUBSTR(字段名, 开始位置, 长度) | 截取指定长度字符串 |
| REPLACE(字段名, 旧字符串, 新字符串) | 替换字符串 |

语句功能：

更新指定表的指定记录。

举例：

将STUDENT表中张三的年龄修改为16。

|  |  |
| --- | --- |
| **UPDATE** | STUDENT |
| **SET** | AGE=16 |
| **WHERE** | NAME=张三 |

2.1 函数修改

KBASE系统在检索语句的UPDATE子句中，还支持其他特定的表达式函数，以实现特定的功能。

#### 2.1.1 函数GETDATE

用法：

GETDATE()

说明：

返回当前的日期。

举例：

将篇名出现了“转换器”的更新日期修改成当前日期。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 更新日期 = getdate() |
| **WHERE** | 篇名 = '转换器' |

#### 2.1.2 函数DATEADD

用法：

DATEADD(interval, number, date)

说明：

返回指定时间间隔的日期。

1. interval： 表示时间类型，设定值如下：

|  |  |  |
| --- | --- | --- |
| 名称 | 简称 | 说明 |
| Year | yy | 年 1753 ~ 9999 |
| Quarter | qq | 季度 |
| Month | mn | 月1 ~ 12 |
| Day | dd | 日，1-31 |
| Week | wk | 周，一年中的第几周，0 ~ 51 |

1. number： 表示时间间隔。正数表示加上对应时间间隔，负数表示减去对应时间间隔。
2. date： 日期或者表达式

举例：

将篇名出现了“转换器”的更新日期修改成在当前日期基础上加一天。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 更新日期 = dateadd("day",1, GETDATE()) |
| **WHERE** | 篇名 = '转换器' |

#### 2.1.3 函数UPPER

用法：

UPPER(字段名)

说明：

将字符串转换成大写。

举例：

将篇名出现了“转换器”的引文字段中的小写字符串转换成大写。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 引文=UPPER(引文) |
| **WHERE** | 篇名 = '转换器' |

#### 2.1.4 函数LOWER

用法：

LOWER(字段名)

说明：

将字符串转换成小写。

举例：

将篇名出现了“转换器”的引文字段中的大写字符串转换成小写。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 引文=LOWER(引文) |
| **WHERE** | 篇名 = '转换器' |

#### 2.1.5函数LEFT

用法：

LEFT(字段名, 长度)

说明：

将字符串的左边指定长度的字节截取返回。

举例：

修改篇名中出现“转换器”的“引文”字段，取左边13个字符长度的字符串。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 引文=LEFT(引文, 13) |
| **WHERE** | 篇名 = '转换器' |

#### 2.1.6函数RIGHT

用法：

RIGHT(字段名, 长度)

说明：

将字符串的右边指定长度的字节截取返回。

举例：

修改篇名中出现“转换器”的“引文”字段，取右边13个字符长度的字符串。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 引文=RIGHT(引文, 13) |
| **WHERE** | 篇名 = '转换器' |

#### 2.1.7函数LTRIM

用法一：

LTRIM(字段名)

说明：

去除字符串前面的空格。

举例：

修改篇名中出现“转换器”的“引文”字段，去掉将该字段值前面的空格。即字段值” The Research and Design of Modulator for Oversampled Audio D/A Converter” 修改成为”The Research and Design of Modulator for Oversampled Audio D/A Converter”。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 引文=LTRIM(引文) |
| **WHERE** | 篇名 = '转换器' |

用法二：

LTRIM(字段名, 字符串)

说明：

去除字符串前面的指定字符，且字符区分大小写。

举例：

修改篇名中出现“转换器”的“引文”字段，去掉该字段值前面的字符串’The’。即字段值” The Research and Design of Modulator for Oversampled Audio D/A Converter” 修改成为” Research and Design of Modulator for Oversampled Audio D/A Converter”。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 引文=LTRIM(引文,‘The') |
| **WHERE** | 篇名 = '转换器' |

#### 2.1.8函数RTRIM

用法一：

RTRIM(字段名)

说明：

去除字符串后面的空格。

用法二：

RTRIM(字段名, 字符串)

说明：

去除字符串后面的指定字符，且字符区分大小写。

举例：

修改篇名中出现“转换器”的“引文”字段，去掉该字段值后面的字符串’ Converter'。即字段值” The Research and Design of Modulator for Oversampled Audio D/A Converter” 修改成为”The Research and Design of Modulator for Oversampled Audio D/A ”。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 引文=LTRIM(引文,‘Converter') |
| **WHERE** | 篇名 = '转换器' |

#### 2.1.9函数TRIM

用法一：

TRIM(字段名)

说明：

去除字符串前后空格。

用法二：

TRIM(字段名, 字符串)

说明：

去除字符串前后指定字符。

#### 2.1.10函数SUBSTR

用法一：

SUBSTR(字段名, 开始位置, 长度)

说明：

从字符串中指定开始位置截取指定长度字符串返回，且开始位置从0开始。

举例：

修改篇名中出现“转换器”的“光盘号”字段，取其第3个字符起截取2位的字符串作为新的光盘号。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 光盘号 = substr(光盘号,2,2) |
| **WHERE** | 篇名 = '转换器' |

#### 2.1.11函数REPLACE

用法一：

REPLACE(字段名, 旧字符串, 新字符串)

说明：

把字段中指定的字符串替换成新字符串。

举例：

修改篇名中出现“转换器”的“引文”字段，将'The Research and Design'替换成'A and B'。

|  |  |
| --- | --- |
| **UPDATE** | cjfd02 |
| **SET** | 引文 = REPLACE(引文,'The Research and Design','A and B') |
| **WHERE** | 篇名 = '转换器' |

3、删除记录

语法格式：

DELETE FROM <表名>

[<WHERE 子句>]

<WHERE 子句>语法同SELECT 语句。

语句功能：

删除指定表的指定记录。

举例：

删除STUDENT表中姓名为“张三”的记录。

|  |  |
| --- | --- |
| **DELETE FROM** | STUDENT |
| **WHERE** | NAME=张三 |

4、重整表

语法格式：

PACK TABLE <表名>

语句功能：

重整指定的表，重整表的目的是为了提高表的存储空间利用率，并可以提高表的访问效率。

重整表过程中，完成整表索引后，系统自动刷新表排序文件。

举例：

重整表CJFD2004。

|  |  |
| --- | --- |
| **PACK TABLE** | CJFD2004 |

5、清空表

语法格式：

CLEAR TABLE <表名>

参数：

<表名> 待清空的表名。

语句功能：

清空一个表中的所有数据，包括索引。

使用说明

无。

举例：

清空表CJFD2004。

|  |  |
| --- | --- |
| **CLEAR TABLE** | CJFD2004 |

6、文档存储类型管理

6.1 创建文本抽取器

语法格式：

CREATE TEXTEXTRACTOR <文本抽取器名>

PIPENAME <管道名>

[MODULENAME <模块名>]

ON <文件扩展名>

6.2 删除文本抽取器

语法格式：

DROP TEXTEXTRACTOR <文本抽取器名>

6.3 文档型数据插入格式

文档型数据对应的原始文件，保存在文件系统中，KBase 的文档型数据类型，只在数据库中保存文档的文件路径等信息，或保存通过文本抽取器抽取的文档文本内容。

文档文件路径相关信息格式为：

<文件名>#<文本抽取器名>

通过这种格式，我们可以像其他数据一样写入文档数据。

第七章 KSQL统一扩展管理语句

针对 KBASE 的功能接口需要，KSQL 对SQL做了大量扩展，这些扩展的语句，都统一以 DBUM 开头，在前面的章节中已经出现过。本章就其他扩展管理语句进行说明。

1、排序词典与排序索引管理

Kbase 中提供了对海量数据进行快速排序和分组的机制。进行这种排序之前，必须对数据根据排序词典建立排序索引。在排序索引建立之后，才可以利用排序索引进行排序和分组。

KBase系统支持两种排序词典，系统排序词典和用户排序词典。系统排序词典主要处理可以预定义排序范围的数据类型，有 INTEGER、DATE等类型。用户排序词典则需要用户来建立。

根据排序词典，就可以给一个表建立排序索引，排序索引存储在扩展名为 .NST的文件中。如cjfd0608的排序索引存储在cjfd0608.nst文件中。

1.1排序词典管理

#### **1.1.1创建用户排序词典**

CREATE SORTDICT<词典名>  
 PATH <存储目录>  
 ALIASNAME <词典名>  
 ORDER BY <SPELL|STROKE|CODE|UNICODE|Formula( <公式> )>  
 MAXWORDLEN <词条最大长度>  
 SORTVALUENUM <排序值个数>

说明：

<词条最大长度>参数范围[0，48]字节，默认值0表示系统支持的最大长度。词典中词条是根据设置的词条最大长度对加入的词条从第一个非空字符开始进行截断处理生成。

<排序值个数>参数范围[1，16]，其中设置成1则该词典适用于单值字段建立排序索引，设置成大于1则该词典适用于多值字段建立排序索引。

<公式> 得到的结果必须为32位整数。

KBase 支持的用于定义排序词典的函数有：

|  |  |  |
| --- | --- | --- |
| **函数名** | **函数的C风格声明** | **说明** |
| KEY | char\* KEY() | 返回待处理的列值，返回值为字符串 |
| ATOI | int ATOI(char \*str) | 将字符串参数str转换为32位整数返回 |
| LEFT | char\* LEFT(char\* str,int len) | 将字符串参数str的左len字节截取返回 |
| RIGHT | char\* RIGHT(char\* str,int len) | 将字符串参数str的右len字节截取返回 |
| SUBSTR | char\* SUBSTR(char\* str,int pos,int len) | 将字符串参数str的第pos开始的len字节截取返回 |
| TRIM | char\* TRIM(char\* str,char c) | 将字符串参数str的两端字符c去掉后返回 |
| TRIM | char\* TRIM(char\* str) | 将字符串参数str的两端空格字符去掉后返回 |
| RTRIM | char\* RTRIM(char\* str,char c) | 将字符串参数str的右端字符c去掉后返回 |
| RTRIM | char\* RTRIM(char\* str) | 将字符串参数str的右端空格字符去掉后返回 |
| LTRIM | char\* LTRIM(char\* str,char c) | 将字符串参数str的左端字符c去掉后返回 |
| LTRIM | char\* LTRIM(char\* str) | 将字符串参数str的左端空格字符去掉后返回 |

举例：

截取表CJFD01其中CN 字段值第3位到第6位字符建立排序索引。

利用公式创建词典：

|  |  |
| --- | --- |
| **CREATE SORTDICT** | CN |
| **PATH** | 'E:\SORTDICT' |
| **ALIASNAME** | ‘’ |
| **ORDER BY** | Formula(ATOI(SUBSTR(KEY(),3,4))) |
| **MAXWORDLEN** | 8 |
| **SORTVALUENUM** | 1 |

创建排序索引：

|  |  |
| --- | --- |
| **DBUM MAKE SORTCOL BY** | CN (CJFD01.CN) |

刷新排序索引文件：

|  |  |
| --- | --- |
| **DBUM REFRESH SORTFILE OF TABLE** | CJFD01 |

#### **1.1.2引入排序词典**

将排序词典文件，引入到数据库管理系统中。

DBUM IMPORT SORTDICT <存储目录+词典名>

举例：

将“作者”的排序词典文件，引入到数据库管理系统中。

|  |  |
| --- | --- |
| **DBUM IMPORT SORTDICT** | 'E:\排序词典\DICT\_SORT\作者.sysdict' |

#### **1.1.3导入导出排序词典文本**

(1) 导出排序词典文本。

将排序词典中的词条，及排序值导出保存到指定的文本文件中。

DBUM EXPORT

SORTDICT <词典名><词典文本文件名>

举例：

将“作者”的排序词典中的词条，及排序值导出保存到指定的文本文件中。

|  |  |
| --- | --- |
| **DBUM EXPORT SORTDICT** | '作者' 'E:\tmp\作者.txt' |

(2) 导入排序词典文本。

将保存在文本文件中的排序词条，直接导入到排序词典中，以便快速重建排序词典。导入后，可能存储排序值冲突，需要刷新或压缩排序词典。

DBUM LOAD

SORTDICT <词典名><词典文本文件名>

举例：

创建词典名为“作者”的排序词典，存储在'E:\tmp'路径下，词条按拼音顺序排序，最大长度是32。

|  |  |
| --- | --- |
| **CREATE SORTDICT** | '作者' |
| **PATH** | 'E:\tmp' |
| **ALIASNAME** | '作者' |
| **ORDER BY** | SPELL |
| **MAXWORDLEN** | 32 |
| **SORTVALUENUM** | 8 |

将保存在文本文件“作者.txt”中的排序词条，直接导入到“作者”的排序词典中。

|  |  |
| --- | --- |
| **DBUM LOAD SORTDICT** | '作者' 'E:\tmp\作者.txt' |

#### **1.1.4刷新与压缩排序词典**

制作排序索引时，为了以后能够正常更新排序索引，在缺省的情况下Kbase按照排序词典词条数为几千万以上构建排序索引，因此排序索引占用较大的空间。如果我们预先知道某些排序词典今后不会再添加大量的词条，则可以对排序词典进行压缩。如果利用压缩后的排序词典重新制作排序索引，将会大大减小排序索引所耗的空间。

|  |
| --- |
| **注意：**如果.nst文件中已经包含某列的未经压缩的排序索引，必须在关闭kbase后，将.nst文件删除，然后重新启动kbase，并使用Dbum make sortcol语句才能压缩该列的排序索引所占空间。 |

刷新词典语法：

DBUM REFRESH SORTDICT <词典名> UNCOMPRESS

压缩词典存储语法：

DBUM REFRESH SORTDICT <词典名> COMPRESS

1.2表排序文件管理

#### **1.2.1定义排序索引**

利用排序词典建立排序索引：

DBUM MAKE SORTCOL

BY <词典名> (表名.列名[,<表名.列名>…])

DBUM MAKE SORTFILE

BY <词典名> (表名.列名[,<表名.列名>…])

例如：

|  |  |
| --- | --- |
| **DBUM MAKE SORTCOL BY** | 来源名称 (CJFD2004.中文刊名) |

|  |  |
| --- | --- |
| **DBUM MAKE SORTCOL BY** | ‘DATE’ (CJFD2004.发表时间) |

注意:

在利用DATE词典制作、刷新排序索引、分组和排序时，必须用英文的引号将其引起来。若 .NST 文件中已经有了该列的排序索引，该语句不会执行任何操作。

#### **1.2.2刷新排序索引**

当表中数据有所变化时，需要重新制作排序索引。

语法格式：

DBUM REFRESH SORTFILE OF TABLE 表名

DBUM REFRESH SORTCOL BY <词典名> (表名.列名[,<表名.列名>…])

增量更新排序文件所有列或指定的列，注意：不是所有的版本都支持此命令，不建议使用**（该功能暂未实现）**。

DBUM SYNCHRONIZE SORTFILE OF TABLE <表名>

DBUM SYNCHRONIZE SORTCOL <列名> OF TABLE <表名>

例如：

|  |  |
| --- | --- |
| **DBUM REFRESH SORTCOL BY** | 来源名称 (CJFD2004.中文刊名) |

注意:

若 .NST 文件中已经有了该列的排序索引，该语句将重新计算排序索引。如果排序索引文件中没有该列的排序索引，则不执行任何操作。

#### **1.2.3管理排序索引文件**

清除排序索引文件中的数据，保留物理排序索引文件：

DBUM CLEAR SORTFILE OF TABLE <表名>

彻底删除排序索引文件：

DBUM REMOVE SORTFILE OF TABLE <表名>

删除特定的排序索引列**（该功能暂未实现）**：

DBUM REMOVE SORTCOL BY <词典名> ( <表名.列名> [,<表名.列名>…] )

同步并行表的排序索引文件，只同步nst文件内容，对于词典不做同步。命令如下：

DBUM UPDATE SORTFILE OF TABLE <表名>

查看排序文件中有哪些排序索引：

SELECT \* FROM sys\_sortcol WHERE tablename=<表名>

1.3表排序文件同步

本地KBase使用远程表数据，当排序列发生变化后，9.0版本以后，支持自动同步排序文件。系统默认30分钟内同步远程表的排序文件到本地KBase服务器上。如果想修改默认同步时间，修改系统参数 TELEAUTOSYNCSORTFILE即可。

系统也支持手工设置同步特定表的排序文件的间隔间隔，手工设置的方法如下：

UPDATE sys\_hotstar\_system

SET querycache=<时间（秒）>

WHERE tablename=<表名>

例如：

如果希望每隔 5 秒同步一次 CJFDTEMP表的排序文件，可以使用如下指令。

|  |  |
| --- | --- |
| **UPDATE** | sys\_hotstar\_system |
| **SET** | querycache=5 |
| **WHERE** | tablename=cjfdtemp |

2、大内存文件缓存

KBase系统通过使用内存对各种文件进行缓存来改善IO性能，进而提升系统整体性能。

KBase支持地址窗口化扩展插件 (AWE)，从而允许在 32 位版本的 Microsoft Windows 操作系统上使用 4 GB 以上的物理内存。最多可支持 64 GB 的物理内存。

KBase 的64位版本，可以更有效的对大内存提供支持，其支持的内存容量仅受操作系统限制。

KBase启用AWE功能之前，需要在所使用服务器上需要配置本地安全设置，在Windows 系统[管理工具]-[本地安全设置]-[用户权限分配]中，为 内存中锁定页面 策略添加Administrator 和system两个用户。

KBase启用大内存来支持个性化文件缓存功能，需要修改配置参数：MEMAWE, 该参数为0时，本功能不启用。

我们可以个性化定义每个文件的内存缓存用量、方法，配置信息可以通过 配置文件AWE.CFG 来指定。该文件存放在KBase安装目录system文件夹下。

配置文件 AWE.CFG 可以手动创建，也可以通过热点统计分析工具支持自动创建，以下分别介绍两种创方法：

2.1手动配置

#### **2.1.1 文件格式**

文件内容格式如下：

[BY FILETYPE]

格式:<文件类型> <相对于标准文件缓冲区大小的倍数>

[BY FILENAME]

<文件名>.<扩展名><块个数>

说明：

[BY FILETYPE]：指定各类文件的最大AWE缓存大小

[BY FILENAME]：指定每个具体文件的最大AWE缓存大小

<块个数> ：指定的块个数为0，则表明没有限制。

举例：指定REC类型文件的最大AWE缓存大小为标准文件缓冲区大小的10倍，且自动加载SCPD.nst、SCSD.nst、SNAD.nst 进行缓存。

首先创建AWE.CFG文件，将该文件存放在KBase安装目录Server/system文件夹下，文件详细内容如下：

[BY FILETYPE]

REC 10

[BY FILENAME]

SCPD.nst 0

SCSD.nst 0

SNAD.nst 0

保存AWE.CFG文件后，需要重启KBase，系统就可以通过使用内存对各种文件进行缓存来改善IO性能。

#### **2.1.2 AWE预加载语句**

##### 2.1.2.1预加载默认空间中文件

语法格式:

DBUM AWECache TABLE <表名><文件扩展名> [FROM <开始块号> [WITH BLOCKCOUNT <块个数>]]

参数:

<表名>预加载的表名。

<文件扩展名>预加载表名的扩展名。

<开始块号> 默认为0，块的大小为4k，缓存指定文件的BLOCKCOUNT个块(默认为到文件尾)

举例：

预加载CJFD2004.nst文件所有块。

|  |  |
| --- | --- |
| **DBUM AWECache TABLE** | CJFD2004 nst |

|  |
| --- |
| **注意：**后缀名为.tagprn 和.tfprn 的两个文件，预加载时只需执行一条命令，语法如下：  DBUM AWECache TABLE <文件名> se |

##### 2.1.2.2预加载存储空间中文件

语法格式:

DBUM AWECache TABLE <表名>STORAGESPACE<存储空间名><文件扩展名> [FROM <开始块号> [WITH BLOCKCOUNT <块个数>]]

参数:

<表名>预加载的表名。

<存储空间名>预加载存储空间的名称。

<文件扩展名>预加载存储空间的扩展名。

<开始块号> 默认为0，块的大小为4k，缓存指定文件的BLOCKCOUNT个块(默认为到文件尾)

举例：

预加载CJFD2004\_频次.rec文件所有块。

|  |  |
| --- | --- |
| **DBUM AWECache TABLE** | CJFD2004 |
| **STORAGESPACE** | 频次 rec |

##### 2.1.2.3预加载索引空间中文件

语法格式:

DBUM AWECache TABLE <表名>INDEXSPACE<索引空间名><文件扩展名> [FROM <开始块号> [WITH BLOCKCOUNT <块个数>]]

参数:

<表名>预加载的表名。

<索引空间名>预加载索引空间的名称。

<文件扩展名>预加载索引空间的扩展名。

<开始块号> 默认为0，块的大小为4k，缓存指定文件的BLOCKCOUNT个块(默认为到文件尾)

举例：

预加载CJFD2004\_频次.idx文件所有块。

|  |  |
| --- | --- |
| **DBUM AWECache TABLE** | CJFD2004 |
| **INDEXSPACE** | 频次 idx |

##### 2.1.2.4预加载词典文件

语法格式:

DBUM AWECache SORTDICT<词典名>[FROM <开始块号> [WITH BLOCKCOUNT <块个数>]]

参数:

<词典名>预加载词典名的名称。

举例：

预加载 关键词.sysdict词典所有块。

|  |  |
| --- | --- |
| **DBUM AWECache SORTDICT** | '关键词' |

以上介绍了手动配置AWE缓存的使用方法，下面做详细举例：

举例：为提升CJFD2004表分组性能，自动加载CJFD2004.NST文件进行缓存，并提升刷新排序索引速度，可预加载所使用的词典文件进行缓存。

执行步骤：

1) 编写配置文件，文件名为AWE.CFG文件，存放于KBase安装目录Server/system文件夹下，加载CJFD2004.NST文件进行缓存，文件内容如下：

[BY FILENAME]

CJFD2004.nst 0

AWE.CFG文件保存后，需要重启KBase服务。

2) 预加载 关键词.sysdict词典所有块，QBE执行如下语句：

|  |  |
| --- | --- |
| **DBUM AWECache SORTDICT** | '关键词' |

注意：词典文件块的大小也需要以4K为单位。

3) 此后执行刷新排序列索引操作及分组操作，可改善IO性能，提升速度。

2.2 使用热点统计分析工具自动配置

使用热点统计分析工具配置AWE过程中用到的语句语法如下：

1. 开启文件访问日志功能语句：

DBUM ENABLE BLOCKLOGGER

1. 关闭文件访问日志功能语句：

DBUM DISABLE BLOCKLOGGER

1. 在客户端执行热点分析程序，自动生成AWE配置文件语句：

DBUM EXECUTE HOTANALYSIS

1. KBase启动时自动加载热点文件语句：

DBUM set AUTOLOADHOTSPOT 1

1. KBase启动时不加载热点文件语句：

DBUM set AUTOLOADHOTSPOT 0

1. 设置热点统计文件个数语句：

DBUM SET BLOCKACCESSFILECNT<文件个数>

<文件个数>将要生成热点文件个数，为大于0的整数。

1. 设置每个热点统计文件大小：

DBUM SET BLOCKACCESSFILESIZE<文件大小>

<文件大小>每个热点文件最大的文件大小，单位为M。

1. 设置热点日志文件打开语句：

DBUM SET BLOCKACCESSFLAG0/1

1表示开启热点日志文件，0表示关闭热点日志文件。

1. 设置每个热点统计文件大小：

DBUM SET BLOCKACCESSBUFSIZE<文件缓存块大小>

<文件缓存块大小>设置热点文件缓存块的大小，单位为M。

以上这些参数值也可以在kbase安装目录server\system下的hotstar.cfg中直接修改配置。

使用热点统计功能自动生成配置文件和预加载热点块操作步骤如下：

1. 在客户端控制开启【文件块访问日志】功能，应用运行一段时间同时工具自动记录热点文件日志，日志默认存放在KBase安装目录server\system文件夹下的block\_access文件中。

在客户端QBE中执行下如下语句，开启【文件访问日志】功能：

|  |  |
| --- | --- |
| **DBUM ENABLE BLOCKLOGGER** |  |

打开【热点日志文件】功能：

|  |  |
| --- | --- |
| **DBUM SET BLOCKACCESSFLAG** | 1 |

执行语句结束后，需要应用运行一段时间，实现热点统计功能。

1. 关闭【文件块访问日志】功能，在客户端执行【热点分析】程序HotAnalysis.exe，自动生成AWE配置文件。

关闭【文件访问日志】功能：

|  |  |
| --- | --- |
| **DBUM DISABLE BLOCKLOGGER** |  |

在客户端执行热点分析程序，自动生成AWE配置文件语句：

|  |  |
| --- | --- |
| **DBUM EXECUTE HOTANALYSIS** |  |

执行语句结束，将自动生成AWE配置文件。

1. 启动【自动加载热点文件】，KBase启动时自动加载热点文件，至此热点块预加载到内存的操作完成。

启动【自动加载热点文件】：

|  |  |
| --- | --- |
| **DBUM set AUTOLOADHOTSPOT** | 1 |

执行语句结束后，需要重启KBase，自动加载热点文件。

2.3 AWE 参数说明

**参数** MEMAWE**：**

设置最大可用内存数，单位为M。MEMAWE参数默认值为28672，在开启AWE功能时，MEMAWE参数值的设置必须小于系统最大内存。

举例：

32G内存的服务器，设置AWE可用最大内存为28G，语句如下：

|  |  |
| --- | --- |
| **DBUM SET MEMAWE** | 28672 |

2.4 其他参数说明

参数 VIRTUAL\_MEM\_TOP：

系统虚拟内存的上限。

64位系统下，0表示总的物理内存大小。

32位系统下，0表示虚拟内存为2G。

参数MEM\_TOP\_SAFE\_PERCENT、MEM\_TOP\_WARNING\_PERCENT、MEM\_TOP\_ERROR\_PERCENT：

以上三个参数指已用内存占VIRTUAL\_MEM\_TOP的百分比，根据内存占用的百分比程序作出相应的处理，保证系统安全。

参数MEM\_TOTAL、MEM\_AVAILABLE ：

以上两个参数是只读的，记录kbase启动时系统的总内存大小和可用内存大小。

3、检索缓存

检索缓存功能实现了灵活选择对检索结果是否缓存，提高检索效率。系统参数SQL\_CACHE\_TYPE可以控制检索缓存的使用情况。

3.1参数值说明

SQL\_CACHE\_TYPE 参数默认为1，有0、1、2三个参数值，不同参数值下检索结果缓存情况对应下表：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **0** | **1** | **2** |
| 检索语句中带SQL\_CACHE | | 不缓存 | 缓存 | 缓存 |
| 检索语句中带SQL\_NO\_CACHE | | 不缓存 | 不缓存 | 不缓存 |
| 检索语句中不指定 | varselect语句 | 不缓存 | 缓存 | 缓存 |
| 带有group子句的检索语句 | 不缓存 | 缓存 | 缓存 |
| 其他检索语句 | 不缓存 | 不缓存 | 缓存 |

3.2参数值设置

**参数**SQL\_CACHE\_TYPE：

参数设置缓存类型，参数值0|1|2。

举例：

设置缓存类型为2

|  |  |
| --- | --- |
| **DBUM SET** |  |
| **SQL\_CACHE\_TYPE** | 2 |

4、集群

4.1 集群角色介绍

集群由中心节点和数据节点两种角色组成。

集群中心节点负责收集集群中表的分布信息；负责监测数据节点有效性；平衡调度任务；记录集群运行日志及性能分析。

数据节点又分为数据检索节点和数据卫星节点，数据检索节点提供数据库管理功能，连接远程表；并负责向中心节点注册有哪些本地表。数据卫星节点只负责本地数据表服务。

4.2集群参数配置

1．参数名：CLUSTERENABLE<ON|OFF>

参数说明：设置集群功能是否启动

设置语句：DBUM set CLUSTERENABLE ON|OFF

2.参数名：CLUSTERCENTERNODE<ON|OFF>

参数说明：设置节点是否为中心管理节点

设置语句：DBUM set CLUSTERCENTERNODE ON|OFF

3．参数名：CLUSTERNODE<ON|OFF>

参数说明：设置节点是否为数据节点

设置语句：DBUM set CLUSTERNODE ON|OFF

4．参数名：CLUSTERCENTERIP<IP>、CLUSTERCENTERPORT<N>

参数说明：指定节点将要参与的集群中心管理节点IP(CLUSTERCENTERIP)和端口号（CLUSTERCENTERPORT）

举例：

|  |  |
| --- | --- |
| **DBUM set** |  |
| **CLUSTERCENTERIP** | 192.168.11.101 |

5．参数名：CLUSTERREGINTERVAL<N>

参数说明：设置集群数据节点注册间隔时间，单位秒

6．参数名：CLUSTERINVALIDINTERVAL<N>

参数说明：设置集群数据节点失效间隔时间，单位秒

7．参数名：CLUSTERLOG<0|1|2>

参数说明：设置集群性能日志选项

8．参数名：CLUSTERMAXWEIGHT<N>、CLUSTERMINWEIGHT<N>

参数说明：设置集群调度最大最小权重

5、其他杂项

5.1 系统变量设置

语法格式：

DBUM SET <变量名><变量值>

例如常用的：

a) DBUM SET SQLLOGON 0

表示将变量SQLLOGON 设为 0，即不记录SQL日志

b) DBUM SET MAXTHREADNUM 200

将系统线程数设置为 200

5.2 个性化设置表的分词方法

当数据库中表的分词参数和本地KBase系统配置中参数不一致时，KBase 9.0开始，支持个性化设置表的分词方法。

系统参数NLPEENABLE 的设置可以控制系统级选用的分词方法。

**系统配置参数说明：**

设置语句：

DBUM SET NLPEENABLE <0|2|3>

参数：

NLPEENABLE参数默认为3，有0、2、3三个参数值，不同参数值对应参数说明见下表：

|  |  |  |
| --- | --- | --- |
| NLPEENABLE | 对应参数 | 参数说明 |
| 0 | WS\_HOTSTAR\_MM | 选用系统默认算法，Hotstar内部基于规则最大长度匹配算法 |
| 2 | WS\_NLPE\_MM | 基于NLPE自然语言处理引擎的最大长度匹配算法 |
| 3 | WS\_NLPE\_FST | 基于NLPE自然语言处理引擎的全切分切词算法 |

数据表也可以支持个性化设置表的分词方法，优先级高于系统参数NLPEENABLE的配置。但需要注意的是，修改表级参数后，需要重建索引，设置的个性化分词方法才能生效。

**个性化设置表的分词方法：**

设置语句：

ALTER TABLE <表名> WORD SEGMENTATION <值>

参数：

<值>::=WS\_DEFAULT|WS\_HOTSTAR\_MM|WS\_NLPE\_MM|WS\_NLPE\_FST

参数WS\_HOTSTAR\_MM|WS\_NLPE\_MM|WS\_NLPE\_FST，不受系统参数NLPEENABLE的限制。通过系统表(sys\_table)可查看个性化设置表的分词属性。

需要注意的是WS\_DEFAULT参数值，将按系统默认参数分词，设置该选项，无法准确辨认该表使用的分词方法，建议尽量不要使用该参数。

举例：

1．修改表CJFD01的NLPEENABLE属性为0，执行语法如下：

ALTER TABLE CJFD01 WORD SEGMENTATION WS\_HOTSTAR\_MM

2．修改表CJFD01的 NLPEENABLE属性为2，执行语法如下：

ALTER TABLE CJFD01 WORD SEGMENTATION WS\_NLPE\_MM

3．修改表CJFD01的 NLPEENABLE属性为3，执行语法如下：

ALTER TABLE CJFD01 WORD SEGMENTATION WS\_NLPE\_FST

4．恢复表CJFD01到系统配置参数的NLPEENABLE值，执行语法如下：

ALTER TABLE CJFD01 WORD SEGMENTATION WS\_DEFAULT

5.3 排序文件同步时间设置

语法格式：

UPDATE SYS\_HOTSTAR\_SYSTEM

SET QUERYCACHE=<刷新时间间隔(秒)>

WHERE TABLENAME=<表名>

参数：

<表名> 待修改刷新同步文件时间间隔的表名。

<刷新时间间隔>设置时间间隔参数，为整数，单位为秒。

语句功能：

手工指定特定远程表的排序文件同步时间间隔，排序列更新频繁的远程表通过这种设置方法，达到及时同步排序文件的效果。

5.4 表读写分离应用设置

支持表读写分离功能的KBase版本，安装后在系统数据库SYSTEM库会存在一张名为SYS\_PREPUB\_INFO的系统表，通过设置该表字段值完成报纸预出版的数据库配置。

该系统表结构如下：

|  |  |
| --- | --- |
| **CREATE TABLE** | SYS\_PREPUB\_INFO |
| **AT** | SYSTEM （ |
| **word segmentation** | TABLENAME VCHAR(255) ASCII INDEX CHAR NORMAL, |
|  | KEYFIELD VCHAR(255) ASCII INDEX CHAR NON, |
|  | COMMITTABLE VCHAR(255) ASCII INDEX CHAR NON, |
|  | ASSISTTABLE VCHAR(255) ASCII INDEX CHAR NON, |
|  | COMMITTIME INTEGER(12) ASCII INDEX INTEGER NON, |
|  | MERGETIME CHAR(24) ASCII INDEX CHAR NON, |
|  | TABLEPATH VCHAR(255) ASCII INDEX CHAR NON) |
|  |  |

其中：

|  |  |
| --- | --- |
| **字段名** | **用途** |
| TABLENAME | 指定日更新表的表名。 |
| KEYFIELD | 表间合并时用的主键字段名称，用来唯一标识记录。 |
| COMMITTABLE | 指定接收数据的临时表表名称。 |
| ASSISTTABLE | 指定临时检索记录的表名称。 |
| COMMITTIME | 指定记录从COMMITTABLE转移到ASSISTTABLE的时间间隔，单位为秒。 |
| MERGETIME | 指定记录从ASSISTTABLE合并到TABLENAME表的物理时刻，格式为hhmmss。 |
| TABLEPATH | 指定COMMITTABLE和ASSISTTABLE表的存放路径。 |

举例：

设置日更新表名为CCNDPREP，主键字段名为“文件名”，临时接收数据的表名为CCND\_COMMIT，提供快速检索的表名为CCND\_DAY，记录从COMMITTABLE转移到ASSISTTABLE表的间隔为120秒，ASSISTTABLE表记录合并到日更新的时刻为凌晨3点，临时表存放在'D:\ccnd\temp'下，INSERT语句如下：

|  |  |
| --- | --- |
| **INSERT INTO** | SYS\_PREPUB\_INFO（ |
|  | TABLENAME,KEYFIELD, |
|  | COMMITTABLE,ASSISTTABLE |
|  | ,COMMITTIME,MERGETIME,TABLEPATH) |
| **VALUES** | (CCNDPREP,文件名, |
|  | CCND\_COMMIT,CCND\_DAY, |
|  | 120,030000,'D:\ccnd\temp') |

设置完成后，重新启动KBase，系统自动生成表结构同CCNDPREP表的CCND\_COMMIT，CCND\_COMMIT\_TMP，CCND\_DAY，CCND\_DAY\_TMP。

5.5 数据排重

语法格式：

DBUM TABLE <表名>

distinct <主键名>

[from <参考表1> [,<参考表2>,...]

[by hash|by index]

[[retain the last| retain the first]

该语句的功能是对表（表名为*<表名>*）按照指定的字段（字段名为*<主键名>*）进行排重。如果希望参考其他表进行排重，就可以增长 from 子句，指定参考的表名列表。

排重算法支持两种，分别是依赖于表的索引（by index）或者需要排重的字段建立哈希表(by hash)进行排重。

排重后，可以保留最前面的记录(retain the first)或者最后(retain the last)的记录。

5.6索引类型相关度范围

KBase系统提供了丰富的数据索引类型，各索引类型的相关度算法也有所不同，可以满足各种应用需求。

* **数值型数据索引类型**

包括INTEGER、QINTEGER、NUM、DATE、QDATE、TIME、INT64，这几种数据类型相关度计算值均为100%。

* **字符串数据索引类型**

包括CHAR、ECHAR、MVCHAR、EMVCHAR，在不同检索特性下，相关度计算值均为100%。而WORD索引类型，相关度计算值在20-40%。

* **文本型数据索引类型**

包括STRING、STRCHAR，在精确与模式检索特性下，相关度计算值范围为30-45%。

MSTRCHAR索引类型，在不同检索特性下，相关度计算值为20-45%。

包括QSTRING、QSTRCHAR、TITLE、EXTITLE、EXTITLECHAR几种索引类型，相关度计算值在1-80%。

包括QTEXT、LFTEXT、TEXT、TEXTCHAR、LTEXT、LTEXTCHAR、ABSTRACT、COMPACTQTEXT、TYTDTEXT、TYTEXT几种索引类型，相关度计算值在1-80%。

* **其他特定用途的数据索引类型**

包括VECTOR、SVSM索引类型，相关度计算值在1-60%。

值得特别指出的是，以上所有数据类型，在KSQL语句的WHERE子句中进行逻辑运算时，相关度计算值会根据各种逻辑组合进行不同的运算。

在进行逻辑“与”运算时，最终的相关度计算值会取两个子值的中间值。在进行逻辑“或”运算时，最终的相关度计算值会高于两个子值，我们视为合理。

第八章 KSQL批量数据处理语句

KBase 主要面向企业市场，它提供多种高效率的批量的数据处理方面，以实现工厂模式下高性能的数据处理机制。

这些数据处理方法包括：

1、排重合并表

语法格式：

BULKLOAD TABLE <表名1>

FROM TABLE <表名2>

WITH KEY = <关键字字段名> ,

MASTER = <表名1或 表名2>

参数：

<表名1> 待合并的表名。

<关键字字段名> 表1和表2中用来唯一 确定记录的字段名称。

语句功能：

把数据表<表名2>中的所有数据追加到数据表<表名1>中，指定字段重复的记录不参加合并，包括索引信息。这两个数据表数据字段结构应该相同。关键字字段应该是表的可查询的字段，能惟一确定记录值。MASTER字段指定重复时要保留表的字段信息。

使用说明：

无。

举例：

合并表 CJFD2003的记录到CJFD2004中，重复时保留CJFD2004中记录。

|  |  |
| --- | --- |
| **BULKLOAD TABLE** | CJFD2004 |
| **FROM TABLE** | CJFD2003 |
| **WITH KEY** | **=**'记录标识' , |
| **MASTER** | **=**CDFD2004 |

2、系统变量设置

语法格式：

DBUM SET RECPARAM<参数>

参数：

<参数>格式为：xxxxB ，其中x表示0/1。

<参数>包含5种情况：

00B：导入/导出REC文件时，转义符无效；导出REC文件同时，不生成INI配置文件；导出REC文件时，只输出第一条记录中的空值字段。

01B：导入/导出REC文件时，转义符无效；导出REC文件同时，生成INI配置文件。

10B：导入/导出REC文件时，转义符生效；导出REC文件同时，生成INI配置文件，且配置文件中仅记录ESC=1，没有FIELDLIST描述信息。

11B：导入/导出REC文件时，转义符生效；导出REC文件同时，生成INI配置文件。

100B：导出REC文件时，空值字段全部输出。

使用说明：

在原有批量导入/导出数据的基础上，增加了对REC文件的转义功能。转义字符（在十六进制中表示为1B）作用于REC和=。

参数第一位表示，导出REC文件时，二进制字段是否以文件形式导出。即当参数第一位设置成0时，导出Base64编码的文本形式；设置成1时，导出文件形式。

参数第二位表示，导出REC文件时，空值字段是否全部输出。即当参数第二位设置成0时，只输出第一条记录中的空值字段；设置成1时，空值字段全部输出。

参数第三位表示，导入/导出REC文件时，是否启用转义。即当变量值第三位设置成0时，表示转义符不起作用；设置成1时，转义符生效，如果待导出的字段内容中包含REC和=，那么导出的REC文件里，REC和=前面添加转义符（在十六进制中表示为1B），如果待导入的REC文件里，REC/=前有转义符，则将去掉REC/=前的转义符进行导入。

参数第四位表示，导出REC文件时，是否生成INI配置文件。即当参数第四位设置成0时，不生成INI配置文件；设置成1时，生成INI配置文件，INI配置文件中存放了表中有效字段的配置信息，其中ESC=0表示转义符无效，ESC=1表示转义符生效。特殊情况，当参数为xx10B时，也生成INI文件。

3、批量导入数据

往表中批量导入数据有两种方法，一是通过标准REC文件导入数据，另一种是导入另一张表中的数据。

语法格式一：

BULKLOAD TABLE <表名>

<REC文件名> [FROM <记录号>]

参数：

<表名> 待导入REC记录的表名。

<REC文件名>待导入REC文件路径及文件名。

<记录号> 指定从REC文件中第几条记录开始导入。记录号为REC文件中记录的序号，为整数，从0开始计数，默认为“0”，指从文本中第一条REC记录开始导入。[FROM <记录号> ] 部分可省略，系统将按默认值执行。如果语句中所写记录号大于REC文件中实际记录数，系统将不导入任何记录。

语句功能：

把指定REC文件中的记录导入到表中。执行该语句导入的记录不带索引信息，导入记录时也不会和表中已有记录进行排重处理。

使用说明：

REC文件内容格式必须正确，否则导入不成功。

举例：

将D:\\cjfd2004目录下名为cjfd2004.txt 的REC文件中所有记录导入到表cjfd2004中，执行语句如下：

|  |  |
| --- | --- |
| **BULKLOAD TABLE** | CJFD2004 |
|  | 'D:\\cjfd2004\\cjfd2004.txt' |

把D:\\cjfd2004目录下名为cjfd2004.txt 的REC文件中第10条记录开始到最后一条记录导入到表cjfd2004中，执行语句如下：

|  |  |
| --- | --- |
| **BULKLOAD TABLE** | CJFD2004 |
|  | 'D:\\cjfd2004\\cjfd2004.txt' |
| **FROM** | 10 |

语法格式二：

BULKLOAD TABLE <表名1>

FROM TABLE <表名2>

参数：

<表名1> 待导入记录的表名。

<表名2> 记录来源表的表名。

语句功能：

把指定表中的记录导入到另一表中。执行该语句导入的记录属性同来源表，源表中有索引的记录导入到新表后同样有索引，反之亦然。通过这种方法导入记录时也不会和表中已有记录进行排重处理。

使用说明：

记录来源表和将要导入的表应该为同结构的表。

举例：

将表CJFD2003中所有记录导入到表CJFD2004中，执行语句如下：

|  |  |
| --- | --- |
| **BULKLOAD TABLE** | CJFD2004 |
| **FROM TABLE** | CJFD2003 |

4、批量导出数据

语法格式：

SELECT [<TOP N>] \* /[<字段名1>,<字段名2>,<字段名3>……]

FROM <表名>

[<WHERE 条件>]

[<GROUP BY 子句>

[<ORDER BY 子句>

[<LIMIT F,N>]/[<LIMIT N>]

INTO < “文件名”>

参数：

<TOP N>导出前N条记录。

<字段名1>待导出的表中字段名，可以是多个，“\*”代表导出表中所有字段。

<表名> 待导出数据的来源表表名。

<条件>筛选导出数据的检索条件，不设筛选条件将导出表中所有记录。

<分组>按照指定列分组。

<排序>按照某一字段进行排序，不设排序将按默认格式导出数据。

<LIMIT F,N> 导出F条记录开始的N条记录（注意，F从0开始）。

<LIMIT N>导出前N条记录。

<文件名>指定将导出的REC文件的存放路径和文件名及后缀。格式如：”D:\ccndtemp\ccndprep.txt”。如果指定路径下文件名已存在，导出后将覆盖原来文件。

语句功能：

把单张表中记录导出到文本文件中。支持导出指定的字段，多字段之间用逗号分开；同样支持导出部分记录，通过where 条件筛选，GROUP BY分组和ORDER BY进行排序。支持关键字TOP、LIMIT导出部分记录，且导出记录数不受分页查询窗口大小限制（KBase 每页最大记录数是 512）。

使用说明：

无。

举例：

将文章表中的标题含有“计算机”的文章，按照期刊号对文章表分组，并且按照相关度排序导出，导出到D:\issuetemp下，文件名为issuetemp\_dlrb.txt，执行语句如下：

|  |  |
| --- | --- |
| **SELECT** | \* |
| **FROM** | 文章表 |
| **WHERE** | 标题=计算机 |
| **GROUP BY** | 期刊号 |
| **ORDER BY** | RELEVANT |
| **INTO** | 'D:\issuetemp\issuetemp\_dlrb.txt’ |

将文章表（1000条记录）中的标题含有“计算机”的文章，按照期刊号对文章表分组，并且按照相关度排序，导出第11~710条到D:\issuetemp下，文件名为issuetemp\_dlrb1.txt，执行语句如下：

|  |  |
| --- | --- |
| **SELECT** | 标题,期刊号 |
| **FROM** | 文章表 |
| **WHERE** | 标题=计算机 |
| **GROUP BY** | 期刊号 |
| **ORDER BY** | RELEVANT |
| **LIMIT** | 10,700 |
| **INTO** | 'D:\issuetemp\issuetemp\_dlrb.txt’ |

5、批量更新方法

语法格式:

DBUM UPDATE TABLE <表名>

FROM <文件名>

WITH KEY <主键列>

[ APPENDING ( <列名> [, <列名>…]) ]

参数:

<表名> 待更新的表名。

<文件名> 用于更新的REC格式数据文件。

**<**主键列> 更新的主键列。

<列名> 用户指定的，采用追加方式更新的列。

语句功能：

KBase 系统提供的批量更新方法，是利用REC文件来进行的数据更新模式。在该模式中，我们在REC文件中，提供用于更新的数据。这里的REC文件格式与 KBase 的交换文件REC文件格式完全一样。但对列的出现顺序有要求。这里，对于每条记录，需要指定一个“主键列”，用于定位更新哪条记录，主键值的唯一性由用户保证。“主键列”总是紧随<REC>标记，作为<REC>记录中的第一个列。其他列的值则是用于修改的数据。修改数据采用两种方式，数据替换方式或数据追加方式。

具体哪些列采用追加方式由KSQL语句指定。在KSQL中关键字APPENDING后面列表中指定的列，采用追加方式更新数据，REC记录中出现的其他列，则采用替换方式更新数据。

在当前KBase系统中，只有 LMVCHAR和 MVCHAR数据存储类型支持按值追加方式，其他数据存储类型如果指定按值追加方式更新，数据更新请求将被忽略。

对于按值追加方式进行的列，只有采用MVCHAR索引类型时，系统才可以对其自动索引。如果采用了其他索引类型（包括MSTRCHAR在内），则必须在更新后，手工重新索引后才可以正常检索的追加的数据。

使用说明：

使用该功能的前提条件是存在格式正确的REC文件，具体格式要求参见语句功能部分。

举例：

对CNKI\_DEPT表执行批量排名更新，机构专业代码字段为主键，后台执行如下命令完成对专业文献全国排名、专业核心期刊发文全国排名、专业基金论文全国排名三个字段的批量更新。

step1.为了提高更新速度，先将将要更新的字段索引属性去掉。

|  |  |
| --- | --- |
| **ALTER TABLE** | CNKI\_DEPT |
| **ALTER** | 专业文献全国排名 |
| **AS** | 专业文献全国排名 integer (8) non,专业核心期刊发文全国排名 |
| **AS** | 专业核心期刊发文全国排名 integer (8) non,专业基金论文全国排名 |
| **AS** | 专业基金论文全国排名 integer (8) non |
| **GO** |  |

step2.更新 专业文献全国排名 字段

|  |  |
| --- | --- |
| **DBUM UPDATE TABLE** | CNKI\_DEPT |
| **FROM** | 'Q:\\导航更新txt\\机构排名文本\\专业文献全国排名.txt' |
| **WITH KEY** | 机构专业代码 |
| **GO** |  |

step3.更新 专业核心期刊发文全国排名 字段

|  |  |
| --- | --- |
| **DBUM UPDATE TABLE** | CNKI\_DEPT |
| **FROM** | 'Q:\\导航更新txt\\机构排名文本\\专业核心期刊发文全国排名.txt' |
| **WITH KEY** | 机构专业代码 |
| **GO** |  |

step4.更新专业基金论文全国排名字段

|  |  |
| --- | --- |
| **DBUM UPDATE TABLE** | CNKI\_DEPT |
| **FROM** | 'Q:\\导航更新txt\\机构排名文本\\专业基金论文全国排名.txt' |
| **WITH KEY** | 机构专业代码 |
| **GO** |  |

step5.修改专业文献全国排名、专业核心期刊发文全国排名、专业基金论文全国排名的索引属性

|  |  |
| --- | --- |
| **ALTER TABLE** | CNKI\_DEPT |
| **ALTER** | 专业文献全国排名 |
| **AS** | 专业文献全国排名 integer (8) uniq,专业核心期刊发文全国排名 |
| **AS** | 专业核心期刊发文全国排名 integer (8) uniq,专业基金论文全国排名 |
| **AS** | 专业基金论文全国排名 integer (8) uniq |
| **GO** |  |

step6.索引更新后的字段

|  |  |
| --- | --- |
| **INDEX** | CNKI\_DEPT |
| **ON** | 专业文献全国排名,专业核心期刊发文全国排名,专业基金论文全国排名 |
| **FROM** | 0 |
| **GO** |  |

6、脚本更新方法

语法格式:

DBUM UPDATE TABLE BY <文件名>

参数:

<文件名> 用于控制批量更新的脚本命令文件名。

脚本更新方法，提供更灵活的批量数据更新方法，各种功能可以在脚本命令中指定。所有的脚本更新，一般都要求被更新的表，有一个事实上的主键。我们根据这个主键，提供各种数据更新功能，包括插入数据、更新数据、删除数据。

具体的 脚本命令包括：

6.1 创建优化的主键索引

语法格式:

CREATE OPTIMIZEKEY <唯一字段名> ON TABLE <表名 >

参数:

<唯一字段名> 用于作为主键的字段名。

<表名> 待更新的表名。

只支持在 INTEGER、CHAR索引类型上建立优化的主键索引，其它类型不支持。

比如，我们CNKI数据库的数据更新，一般是根据“文件名”字段来更新的，这个“文件名”字段其类型为CHAR，值唯一，就可以作为优化的主键。

创建优化的主键索引，实际上是基于索引信息，快速的在内存中建立一个主键的映射表，以便大大加快后续更新操作需要频繁访问主键，以优化数据更新的效率。

6.2 释放优化的主键索引

语法格式:

DROP OPTIMIZEKEY <唯一字段名> ON TABLE <表名 >

参数:

<唯一字段名> 用于作为主键的字段名。

<表名> 待更新的表名。

释放内存中建立起来的优化的主键索引相关数据。

6.3 批量插入记录

语法格式一:

INSERT INTO TABLE <表名> WITH KEY = <唯一字段名>

语法格式二:

EXCLUSIVE INSERT INTO TABLE <表名> WITH KEY = <唯一字段名>

语法格式三:

NEW INSERT INTO TABLE <表名> WITH KEY = <唯一字段名>

参数:

<唯一字段名> 用于作为主键的字段名。

<表名> 待更新的表名。

功能：

支持排重的批量插入记录，插入数据的时候，将根据唯一字段名排重，如果唯一字段值重复，则采用更新的方法更新数据，否则是插入数据。

如果采用语法格式一，更新数据时，重复数据的更新方法是，删除旧的重复记录，增加新的记录，这时，如果某些字段没有设定值，这些字段的值将为空。

如果采用语法格式二，更新数据时，重复数据的更新方法是，在旧的记录上修改数据，这时，如果某些字段没有设定值，这些字段的值将不被改变。

如果采用语法格式三，更新数据时，仅插入新的数据，即重复数据不更新、不修改，保留原始记录。

实例：

EXCLUSIVE INSERT 用在CDSUpdate 数据更新中，应该改为INSERT 以提高性能。

6.4 批量修改记录

语法格式:

UPDATE TABLE <表名> WITH KEY = <唯一字段名>

参数:

<唯一字段名> 用于作为主键的字段名。

<表名> 待更新的表名。

功能：

支持的批量修改记录，修改数据的时候，将根据唯一字段名定位记录序号。

6.5 批量删除记录

语法格式:

DELETE FROM TABLE <表名>

WITH KEY = <唯一字段名> ,

RELEVANTCOL=NULL,

RELEVANTINDEX=<相关索引列表>

参数:

<唯一字段名> 用于作为主键的字段名。

<表名> 待更新的表名。

6.6 批量替换记录

语法格式:

REPLACE TABLE <表名> WITH KEY = <唯一字段名>

参数:

<唯一字段名> 用于作为主键的字段名。

<表名> 待更新的表名。

功能:

与 UPDATE 区别在于，可以更新用来标示记录的<唯一字段>。 公司内部没有用到这个特性， 准备去掉这个特性， 改为一个专门修改主键的方法来代替。

6.7 刷新表排序文件

语法格式:

REFRESH SORTFILE OF TABLE <表名>

参数:

<表名> 待刷新排序文件的表名。

功能:

支持刷新表排序文件。该指令可灵活使用于脚本中任意位置，推荐在数据变更完成后使用。

6.8 设置批量处理的数据文件

批量处理数据的格式如下：

INFILE "<文件名>"

这个指令，主要配合前面的这种操作指令，提供操作的数据。

6.9 脚本批量处理实例

针对表CJFDTEMP，完成如下处理：

1. 替换插入CJFD\_A\_INSERT.TXT和CJFD\_B\_INSERT.TXT两个文件的内容，如果文件中主键和表中记录有重复，删除表中原来记录，插入文件中记录；如果文件中主键表中不存在，直接插入记录；
2. 依文件CJFD\_C\_UPDATE.TXT和CJFD\_D\_UPDATE.TXT修改对应主键的特定字段；
3. 依文件CJFD\_A\_DELETE.TXT内容删除表中对应主键所在记录。

执行步骤：

1. 按规格编写需求中提到的文件内容，主键《文件名》索引类型为char，且有索引。文件中每条REC记录的主键紧随<REC>标识。
2. 编写执行文件，文件名为update.txt，存放于E：根目录下，文件内容如下：

|  |  |
| --- | --- |
| **CREATE OPTIMIZEKEY** | 文件名 |
| **ON TABLE** | CJFDTEMP |

|  |  |
| --- | --- |
| **INSERT INTO TABLE** | CJFDTEMP |
| **WITH KEY =** | 文件名 |
| **INFILE** | "E:\\ CJFD\_A\_INSERT.TXT" |
| **INFILE** | "E:\\ CJFD\_B\_INSERT.TXT" |

|  |  |
| --- | --- |
| **UPDATE TABLE** | CJFDTEMP |
| **WITH KEY =** | 文件名 |
| **INFILE** | "E:\\ CJFD\_C\_UPDATE.TXT" |
| **INFILE** | "E:\\ CJFD\_D\_UPDATE.TXT" |

|  |  |
| --- | --- |
| **DELETE FROM TABLE** | CJFDTEMP |
| **WITH KEY =** | 文件名 |
| **INFILE** | "E:\\ CJFD\_A\_DELETE.TXT" |

|  |  |
| --- | --- |
| **DROP OPTIMIZEKEY** | 文件名 |
| **ON TABLE** | CJFDTEMP |

1. 后台执行语句：

|  |  |
| --- | --- |
| **DBUM UPDATE TABLE BY** | "E:\\ UPDATE.TXT" |

第九章 KSQL文本处理语句

1、文本处理 KSQL概述

文本处理是服务器端的智能文本挖掘扩展组件。它主要用来完成对文本信息中知识的挖掘。

而文本处理 KSQL则是驱动这些具体功能的KSQL语句。

由于文本处理属于扩展组件，所以在使用KSQL语句的时候必须要加上使用扩展组件的信息。使用扩展组件的语句是：Use ext <扩展组件名称>。

这里我们使用：Use ext KExt\_STM。

文本处理 KSQL语句是在现有的KSQL语句上扩展的一组用来调用KExt\_STM功能的KSQL语句。

按照文本处理 KSQL的形式，我们可以把其分成两类。

第一类文本处理 KSQL主要用来用于学习指令或者预处理指令，该系列语句一般是对数据做一个预先的处理，对后续的处理做一个准备。该系列语句以CREATE关键字开头，并且不携带返回数据，通过语句执行的返回值来判断语句是否执行成功。

此类文本处理 KSQL语句会有一个From关键字用来指定输入的数据。输入数据源可以从表，文件夹或者由其两者同时输入。当数据从表中输入中，可以在输入表的字段上设置相应的输入权值。权值越大，表示相对的信息越重要。当信息从文件夹输入时，程序会递归遍历文件夹下所有的文件，即包括子文件夹下所有的文件。

第二类文本处理 KSQL语句多用于处理数据，这类指令不仅处理数据，同时在处理的过程中还会生成处理结果。所以在语句中需要指定保存输出结果的表和字段。指令执行完毕之后，用户可以使用查看相应表信息来获取结果信息。

此类语句也有一个From关键字用来指定输入的数据。输入数据源可以是表或者文件夹。当输入源从表输入时，除了可以指定字段的权值，用户还可以指定一个表输入的范围，对于输入的范围的起始和结束都是闭区间，同时起始行的位置是从0开始。当信息从文件夹输入时，程序会递归遍历文件夹下所有的文件。

譬如指定数据从表中输入，

From Table = CJFD1999 with (篇名,10)(摘要,25)(全文,1)

表示从表CJFD1999中取出“篇名”“摘要”“全文”这三个字段作为输入。后面的数字代表该字段在输入中所占的权重值。该值越大，表示相对的信息越重要。通常的取值在1~100之间。在上例中，“篇名”“摘要”“全文”的权重分别是10，25，1。这表示在所有的输入中，摘要字段的信息最重要，最能代表输入文本的特征。全文的信息相对则不太重要。

支持表中指定范围输入的语句：

From Table = CJFD1999 with (篇名,10)(摘要,25)(全文,1)

StartRec = 0 EndRec = 99

表示处理CJFD1999的前100条语句。

而 From File = "C:\文本信息"

则表示把"C:\文本信息"目录下的所有文件递归进行输入。

对于有些语句来说，当数据来源于KBASE数据库中的时候，参数设置中需要有一项TextSource=Table，同时根据用户输出位置的不同，参数设置中还包括一项OutputSrcTable参数。OutputSrcTable表示是否将结果输出到当前处理的表。如果是输出到当前的表中，那么就不需要再指定输出的目的表。

带有输出数据的语句，数据的输出则会写入到语句To所指定数据表和字段中。如果写入的数据表不存在，程序则会创建一个新表。在写入表的时候，除了把结果数据写入到指定的字段下，同时还会根据来源数据的不同，写入相应的数据来源信息。如果处理数据来源于数据表，那么系统会在目的表上加入“表名”和“行号”两个字段。如果处理数据来源于文件，则会在目的表上加入“文件名”字段。如果已经存在名称相同的表，运算结果是以追加的形式写入。

最后，所有的语句都离不开ON CJFD，它表示语句操作所依赖的知识域。CJFD是我们系统中默认的知识域名称，当然也可以指定用户自己创建的知识域名称。注意知识域是大小写相关的。

1、文本挖掘引擎启动

语法格式:

START ENGINE STM

参数:

固定格式，用来启动文本挖掘引擎。

2、文本挖掘引擎结束

语法格式:

STOP ENGINE STM

参数:

固定格式，用来停止文本挖掘引擎。

3、创建知识域

语法格式一：

CREATE KDomain <知识域名称> (<知识域路径>, <词典名称>, <词典路径>,<知识域类型>)

语法格式二：

CREATE KDomain <知识域名称>

参数：

<知识域名称>表示要创建的知识域名称。

<知识域路径>表示创建知识域保存的路径。默认为NULL，表示创建在系统目录的TextMiner子目录下。

<词典名称>表示词典名。默认为NULL，表示使用词典SYS\_GENERAL\_DIC。

<词典路径>表示词典的路径。可以忽略。

<知识域类型>表示所创建知识域的类型，“0”为中文、“1”为英文。

若使用后一种语法格式的话，表示后面四个参数都将使用默认值NULL。

举例：

以词典sys\_small\_dict创建一个名为CJFD2的知识域。

|  |  |
| --- | --- |
| **CREATE KDomain** | CJFD2 |
|  | ("d:\work\bin\textminer","sys\_small\_dict",  "d:\work\bin\directory" ,0) |

以默认参数创建一个名为CJFD3的知识域。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **CREATE KDomina** | CJFD3 |
|  | (NULL, NULL, NULL, 0) |

4、删除知识域

语法格式：

DELETE KDomain <知识域名称>

参数：

<知识域名称>表示要删除的知识域名称。

举例：

删除CJFD2知识域。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **DELETE KDomain** | CJFD2 |

注意：

文本处理模块为扩展模块，执行语句之前必须加上 use ext KExt\_STM。

5、启动知识域

语法格式：

OPEN KDomain <知识域名称>

参数：

<知识域名称>表示要启动的知识域名称。

举例：

启动CJFD2知识域。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **OPEN KDomain** | CJFD2 |

6、关闭知识域

语法格式：

CLOSE KDomain <知识域名称>

参数：

<知识域名称>表示要关闭的知识域名称。

举例：

关闭CJFD2知识域。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **CLOSE KDomain** | CJFD2 |

7、特征词分布规律分析

特征词分布规律分析是产生知识域的基础，通过学习大量指定类别的文本信息，可以产生一定的知识模型，从而为后续的知识挖掘做准备。产生特征词分布规律分析器需要两次操作，第一次是学习，第二次是学习结束。一次完整的操作才能产生一个有效的特征词分布规律分析器。

语法格式一：

use ext KExt\_STM

CREATE WordDistrAnalyst

<From table = <表1[,表2,…]> with <(列1,权重)>[<(列2,权重)>,…]>

startrec =<整数> endrec = <整数>

ON <知识域名称>

语法格式一：

use ext KExt\_STM

CREATE WordDistrAnalyst

<From file = <路径1[,路径2]>>

ON <知识域名称>

GO

特征词分布规律学结束，语法格式：

CREATE WordDistrAnalyst END

SET [Type= <类型>,] [MaxFeatureValue = <阈值>,] [MinFeatureValue = <阈值>]

ON <知识域名称>

参数：

Type为特征提取算法的类型。其值为规定好的字符串集合。默认值为“DF”。各种类型分别表示如下。

|  |  |
| --- | --- |
| **Type 参数** | **含义** |
| DF | 表示使用文频分析算法。 |
| MI | 表示使用互信息分析算法 |
| IG | 表示使用信息增益分析算法 |
| CHI | 表示使用χ2分析算法 |
| MaxFeatureValue | 为最大特征选择阈值。其值为浮点类型。默认值为0.2 |
| MinFeatureValue | 为最小特征选择阈值。其值为浮点类型。默认值为0.0000001。 |

举例：

将库1999，2005，2007，和“C:\文本信息”里面的信息在CJFD 知识域上做一个特征词分布规律的学习，同时最大最小特征选择阈值分别为0.4和0.1，那么可以输入如下语句。

|  |  |
| --- | --- |
| **CREATE WordDistrAnalyst** |  |
| **FROM TABLE =** | CREATE WordDistrAnalyst |
| **FROM TABLE =** | CJFD1999 with (篇名, 10)(摘要,25)(全文,1) |
| **FROM TABLE =** | "C:\医学信息" |
| **ON** | CJFD |

最后，使用结束指令完成整个特征词分布规律的学习过程。

|  |  |
| --- | --- |
| **CREATE WordDistrAnalyst** | END |
| **SET** | Type="DF", MaxFeatureValue=0.4, MinFeatureValue=0.1 |
| **ON** | CJFD |

8、文本分类器

文本分类器是指学习指定类别的文本，产生一个文本分类的模版，依据该模版可以对数据自动分类。文本学习过程主要包括文本学习和文本学习结束两步。

|  |
| --- |
| **注意：**当在指定知识域上使用某种算法学习后，那么其后在该知识域上的学习也必须使用这种算法，否则将会失败。同时注意要使学习生效，最后必须要调用文本分类学习器结束指令。 |

语法格式：

CREATE TextClassifier

SET [Type="KNN|SVM|BAYES|SVD",] [IncrementLearn=<true|false>]

ClassName = "<类型名称>", [UseColValue=<true|false>]

<From table = <表1[,表2,…]> with <(列1,权重)>[<(列2,权重)>,…]>|

startrec = <整数> endrec =<整数>

<From file = <路径1[,路径2]>>

ON <知识域名称>

参数：

Type为文本分类算法的类型。其值为规定好的字符串集合。默认值为“KNN”。各种类型分别表示如下。

|  |  |
| --- | --- |
| **Type 参数** | **含义** |
| KNN | 表示使用K最近邻分类器 |
| SVM | 表示使用支持向量机分类器 |
| BAYES | 表示使用贝页思分类器 |
| ClassName | 为样本学习的类别。其值为能表示输入信息特征的字符串 |
| IncrementLearn | 是一个布尔变量。TRUE表示在现有基础上继续学习，FALSE是销毁现有已经学习好的模型，重新建立一个新的模型。 |
| UseColValue | 是一个布尔变量。TRUE表示使用表字段分类信息，FALSE是使用自定义分类信息。 |

文本分类学习器结束，语法格式：

CREATE TextClassifier end

ON <知识域名称>

参数：

知识域名称为将要学习结束的知识域。

举例：

将库表cjfd1999，cjfd2005，cjfd2007，和“C:\医学”里面的数据在CJFD知识域上学习，同时学习的分类为“图形论文”，那么可以输入如下语句。

|  |  |
| --- | --- |
| **CREATE TextClassifier** |  |
| **SET** | Type="KNN",IncrementLearn=TRUE,ClassName="图形论文", UseColValue=FALSE |
| **FROM TABLE =** | CJFD2005, CJFD2007 with (篇名,15)(中文摘要,20)(全文,1) |
| **FROM TABLE =** | CJFD1999 with (篇名,10)(摘要,25)(全文,1) |
| **Startrec =** | 1 |
| **endrec =** | -1 |
| **From File =** | "C:\医学信息" |
| **ON** | CJFD |
| **GO** |  |

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **Create TextClassifier** | END |
| **ON** | CJFD |

9、信息过滤器

内容信息过滤器是通过用户指定的文本信息，生成一个文本过滤模板，然后用户可以通过信息过滤器来过滤文本信息。

语法格式：

CREATE InfoFilter

SET ProfileName="<模版名称>" [,IncrementLearn=<true|false>]

<From Table = <表1[,表2,…]> with <(列1,权重)>[<(列2,权重)>,…]>

startrec = <整数> endrec =<整数>

<From file = <路径>

ON <知识域名称>

参数：

ProfileName为信息过滤器的模版名称。

IncrementLearn是一个布尔变量。TRUE表示在现有过滤器模版的基础上继续学习。FALSE是清空现有已生成的模版，重新建立新的过滤器模板。

举例：

将库表cjfd2005，cjfd2007，“C:\空气净化信息”中的数据在CJFD知识域上生成名为“环保”的过滤器模版，输入如下语句：

|  |  |
| --- | --- |
| **CREATE InfoFilter** |  |
| **SET** | ProfileName= "环保", IncrementLearn=TRUE |
| **FROM TABLE =** | CJFD2005, CJFD2007 with (篇名,15)(中文摘要,20)(全文,1) |
| **FROM TABLE =** | CJFD1999 with (篇名,10)(摘要,25)(全文,1) |
| **Startrec =** | <起始记录号> |
| **endrec =** | <结束记录号> |
| **From File =** | "C:\空气净化信息" |
| **ON** | CJFD |

10、文本分类

文本分类器就是把当前指定的数据在已经学习好的文本分类器模型上进行分类。在使用文本分类器功能之前必须要调用文本分类学习器。

语法格式：

形式一（分类KBASE数据库中信息）：

TextClassify

SET [Type="KNN|SVM|BAYES|SVD",] [OutputSrcTable=<布尔>]

[TextSource=TABLE,] [ClassCount=<整数>,]

From Table = <表1[,表2,…]> with <(列1,权重)>[ <(列2,权重)>,…]>

[StartRec = <整数> EndRec = <整数> ]

ON CJFD

To [Table = <表名>] Field = <字段名>

参数：

Type为使用何种分类算法，同文本分类学习器。

TextSource为文章来源，TABLE表示来自KBASE数据库。

ClassCount为欲返回的分类数目。默认为1。

举例

将库表cjfd2005，cjfd2007的前3000条数据在知识域CJFD上进行分类，分类的结果保存在表STM\_RESULT的CLASSIFY\_RESULT字段

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **TextClassify** | ProfileName= "环保", IncrementLearn=TRUE |
| **SET** | Type="KNN", TextSource=Table, ClassCount=1, |
| **OutputSrcTable=** | FALSE |
| **From Table =** | CJFD2005, CJFD2007 with (篇名,15)(中文摘要,20)(全文,1) |
| **StartRec=** | 0 |
| **EndRec=** | 3000 |
| **ON** | CJFD |
| **To** | Table= " STM\_RESULT" FIELD="CLASSIFY\_RESULT" |

形式二（分类文件信息）：

TextClassify

SET [Type="KNN|SVM|BAYES|SVD",]

TextSource=FILE [ClassCount=<整数>,]

From File = <路径1[,路径2,…]>

ON CJFD

To Table = <表名> Field = <字段名>

参数：

同上。

举例：

将“C:\文本分类1”和“C:\文本分类2”这两个目录下的文件在知识域CJFD上进行分类，分类的结果保存在表STM\_RESULT的CLASSIFY\_RESULT字段

|  |  |
| --- | --- |
| **TextClassify** |  |
| **SET** | Type="KNN", TextSource=File, ClassCount=1 |
| **From File=** | "C:\文本分类1", "C:\文本分类2" |
| **ON** | CJFD |
| **To Table=** | " STM\_RESULT" FIELD="CLASSIFY\_RESULT" |

11、信息过滤

信息过滤就是把当前指定的数据在已经学习好的文本过滤模型上进行过滤，输出该文本与过滤模型的相似度。在使用文本过滤器功能之前必须要调用文本过滤学习器。

语法格式：

形式一（分类KBASE数据库中信息）：

InfoFilter

SET ProfileName="<模版名称>",OutputSrcTable=<布尔值>

From Table = <表1[,表2,…]> with <(列1,权重)>[ <(列2,权重)>,…]>

[StartRec = <整数> EndRec = <整数> ]

ON CJFD

To [Table = <表名>] Field = <字段名>

参数：

ProfileName为过滤器模版的名称。

举例：

将库2005，2007的数据在知识域CJFD上的过滤模版“学术论文”中进行相似过滤，过滤的结果保存在源表STM\_RESULT的INFOFILTER\_RESULT字段。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **InfoFilter** | Type="KNN", TextSource=File, ClassCount=1 |
| **SET ProfileName=** | "学术论文", |
| **OutputSrcTable=** | TRUE |
| **From Table=** | CJFD2005, CJFD2007 with (篇名,15)(中文摘要,20)(全文,1) |
| **ON** | CJFD |
| **TO** | TABLE="STM\_RESULT" FIELD="INFOFILTER\_RESULT" |

形式二（分类文件信息）：

InfoFilter

SET ProfileName="<模版名称>", TextSource=FILE

From File = <路径1[,路径2,…]>

ON CJFD

To Table = <表名> Field = <字段名>

参数：

同上。

举例：

将“C:\学术论文1”和“C:\学术论文2”这两个目录下的文件在在知识域CJFD上的过滤模版“学术论文”中进行相似过滤，分类的结果保存在表STM\_RESULT的INFOFILTER\_RESULT字段。

|  |  |
| --- | --- |
| **InfoFilter** |  |
| **SET ProfileName=** | "学术论文", |
| **TextSource=** | File |
| **From File=** | " C:\学术论文1", "C:\学术论文2" |
| **ON** | CJFD |
| **TO TABLE=** | "STM\_RESULT" FIELD="INFOFILTER\_RESULT" |

12、相似索引

相似索引用来提取指定文本的VSM向量模型。

语法格式：

形式一（分类KBASE数据库中信息）：

SimIndex

SET MaxFeatureNum=<特征数量>, SortType=<0|1>, OutputWord=<布尔值>,

OutputSrcTable=<布尔值>

From Table = <表1[,表2,…]> with <(列1,权重)>[ <(列2,权重)>,…]>

[StartRec = <整数> EndRec = <整数> ]

ON CJFD

To [Table = <表名>] Field = <字段名>

参数：

MaxFeatureNum为输出的VSM数量，-1为输出所有的VSM模型，否则输出指定数目的VSM模型。如果指定的VSM模型数量大于所有的VSM模型数量，则仅输出所有的VSM模型数量。

SortType为排序类型。0为按权重降序排列，1为按特征标识升序排列。

OutputWord为是否输出相应特征词。

举例：

将库表cjfd2005，cjfd2007的数据在知识域CJFD上生成VSM空间向量，生成的结果保存在STM\_RESULT表的SIM\_RESULT字段中。假设我们只取20个VSM向量并且按照升序排列。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **SimIndex** |  |
| **SET MaxFeatureNum=** | 20 |
| **SortType=** | 1 |
| **OutputWord=** | TRUE |
| **OutputSrcTable=** | FALSE |
| **From Table =** | CJFD2005, CJFD2007 with (篇名,15)(中文摘要,20)(全文,1) |
| **startrec =** | 1 |
| **endrec =** | -1 |
| **ON** | CJFD |
| **TO** | TABLE="STM\_RESULT" FIELD="SIM\_RESULT" |

形式二（分类文件信息）：

SimIndex

SET MaxFeatureNum=<特征数量>, SortType=<0|1>, OutputWord=TRUE, TextSource=File

From File = <路径1[,路径2,…]>

ON CJFD

To Table = <表名> Field = <字段名>

参数：

同上。

举例：

将“C:\文本”这个目录下的文件在知识域CJFD上生成VSM空间向量模型，并将结果保存在STM\_RESULT表的SIM\_RESULT字段中。

use ext KExt\_STM

SimIndexSET MaxFeatureNum=20, SortType=1, OutputWord=FALSE

From File="C:\文本"

ON CJFD

TO TABLE="STM\_RESULT" FIELD="SIM\_RESULT"

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **SimIndex** |  |
| **SET MaxFeatureNum=** | 20 |
| **SortType=** | 1 |
| **OutputWord=** | FALSE |
| **From File=** | "C:\文本" |
| **ON** | CJFD |
| **TO** | TABLE="STM\_RESULT" FIELD="SIM\_RESULT" |

13、文本聚类

文本聚类是指对尚未归类的数据进行自动的分类处理，将相似的数据智能化分，使每个组中的文件互相接近。

语法格式:

形式一（分类KBASE数据库中信息）：

TextCluster

SET [Type="K\_MEANS|AHC",] [ClusterNum=<聚类数>,]

MaxDocsPerCluster=<文档数>,

TextSource=Table, InitFlag=RANDOM,OutputSrcTable=<布尔值>

From Table = <表1[,表2,…]> with <(列1,权重)>[ <(列2,权重)>,…]>

[StartRec = <整数> EndRec = <整数> ]

ON CJFD

To [Table = <表名>] Field = <字段名>

参数:

TYPE为聚类的类型，为以下集合的字符串：

K\_MEANS 表示聚类时使用K\_MEANS扩展算法。默认类型。

AHC表示聚类时使用AHC层次凝聚算法。

ClusterNum为生成的聚类数量，是一个整数，默认为5。

MaxDocsPerCluster为每个聚类中的最大文档数量，默认为200。

举例:

将库表cjfd2005，cjfd2007的数据在知识域CJFD上生成聚类信息，生成的结果保存在表STM\_RESULT的CLUSTER\_RESULT字段中。假设我们将其分成50个聚类。 use ext KExt\_STM

|  |  |
| --- | --- |
| **TextCluster** |  |
| **SET** | type="K\_MEANS", ClusterNum=50, MaxDocsPerCluster=1000, |
| **TextSource=** | Table |
| **OutputSrcTable=** | FALSE |
| **From Table=** | CJFD2005, CJFD2007 with (篇名, 15)(中文摘要,20)(全文,1) |
| **ON** | CJFD |
| **TO** | TABLE="STM\_RESULT" Field="CLUSTER\_RESULT" |

形式二（分类文件信息）：

TextCluster

SET [TYPE="K\_MEANS|AHC",] [ClusterNum=<聚类数>,]

MaxDocsPerCluster=<文档数>,

TextSource=FILE

From File = <路径1[,路径2,…]>

ON CJFD

To Table = <字符串> Field = <字符串>

参数：

同上。

举例：

将“C:\聚类文本1”和“C:\聚类文本2”这两个目录下的文件在在知识域CJFD上生成聚类信息，并将结果保存在表STM\_RESULT的CLUSTER\_RESULT字段字段。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **TextCluster** |  |
| **SET** | type="K\_MEANS", ClusterNum=50, MaxDocsPerCluster=1000, |
| **TextSource=** | FILE |
| **From FILE=** | "C:\聚类文本1", "C:\聚类文本2" |
| **ON** | CJFD |
| **TO** | TABLE="STM\_RESULT" Field="CLUSTER\_RESULT" |

14、关键词提取器

关键词提取是在指定的文本中智能化的提取其中关键的词语。

语法格式：

形式一（分类KBASE数据库中信息）：

KeyWordExtractor

SET [WordNum=<关键词个数>,] ReturnWeight=<布尔值>, ReturnChinese=<布尔值>

TextSource=Table, OutputSrcTable=<布尔值>

From Table = <表1[,表2,…]> with <(列1,权重,[布尔值])>[,…]>

[StartRec = <整数> EndRec = <整数> ]

ON CJFD

To [Table = <表名>] Field = <字段名>

参数：

WordNum 设置抽取的关键词数目。默认为5。

ReturnWeight是否返回关键词的权重。

ReturnChinese是否返回中文关键词，只对英文数据有效。

注意，相比别的语句，关键词在表中输入的时候，除了要指明字段和权重以外，还需要指明处理的时候该字段是否分词，默认为FALSE，即不分词。

举例：

将库表cjfd2005，cjfd2007的数据在知识域CJFD上提取关键词信息，生成的结果保存在表STM\_RESULT的KEYWORD\_RESULT字段中。同时在处理时对于输入的表中“中文摘要”这个字段，不做分词处理，那么相应的语句如下：

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **KeyWordExtractor** |  |
| **SET** | WordNum=5, ReturnWeight=TRUE, |
| **TextSource=** | Table |
| **OutputSrcTable=** | FALSE |
| **From Table=** | CJFD2005, CJFD2007 with (中文摘要,20,true)(全文,1) |
| **ON** | CJFD |
| **TO** | TABLE=STM\_RESULTField= KEYWORD\_RESULT |

形式二（分类文件信息）：

KeyWordExtractor

SET [WordNum=<关键词个数>,]ReturnWeight=<布尔值>,

TextSource= File, ReturnChinese=<布尔值>

From File = <路径1[,路径2,…]>

ON CJFD

To Table = <表名> Field = <字段名>

参数：

同上。

举例：

将“C:\文本”这个目录下的文件在在知识域CJFD上提取关键词信息，生成的结果保存在表STM\_RESULT的KEYWORD\_RESULT字段中。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **KeyWordExtractor** |  |
| **SET** | WordNum=10, ReturnWeight=TRUE, TextSource=FILE |
| **From File=** | "C:\文本" |
| **ON** | CJFD |
| **TO** | TABLE="STM\_RESULT" Field="KEYWORD\_RESULT" |

15、自动摘要提取器

自动摘要提取是在指定的文本中智能化的提取其文本的摘要信息。

语法格式：

形式一（分类KBASE数据库中信息）：

AbstractExtractor

SET SentenceNum=<句子数>,TextSource=Table,SourceField=<表字段值>,

OutputSrcTable=<布尔值>

From Table = <表1[,表2,…]> with <(列1,权重)>[ <(列2,权重)>,…]>

[StartRec = <整数> EndRec = <整数> ]

ON CJFD

To [Table = <表名>] Field = <字段名>

参数：

SentenceNum为自动摘要提取的句子数量。默认为5句。

SourceField为文章正文字段。也就是需要进行提取的文章字段。一般指定“全文”字段。

举例：

将库表cjfd2005，cjfd2007的字段“全文”在知识域CJFD上进行自动摘要的提取，生成的结果保存在表STM\_RESULT的ABSTRACT\_RESULT字段中。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **AbstractExtractor** |  |
| **SET** | SentenceNum=5, TextSource=Table, SourceField=全文 |
| **From Table=** | CJFD2005, CJFD2007 with (篇名,15)(中文摘要,20)(全文,1) |
| **ON** | CJFD |
| **TO** | TABLE="STM\_RESULT" Field="ABSTRACT\_RESULT" |

形式二（分类文件信息）：

AbstractExtractor

SET SentenceNum=<句子数>, TextSource=FILE

From File = <路径1[,路径2,…]>

ON CJFD

To Table = <表名> Field = <字段名>

参数：

同上。

举例：

将“C:\文本”这个目录下的文件在在知识域CJFD上提取关键词信息，生成的结果保存在表STM\_RESULT的ABSTRACT\_RESULT字段中。

|  |  |
| --- | --- |
| **use ext KExt\_STM** |  |
| **AbstractExtractor** |  |
| **SET** | SentenceNum=5, TextSource=FILE |
| **From File=** | "C:\文本" |
| **ON** | CJFD |
| **TO** | TABLE="STM\_RESULT" Field="ABSTRACT\_RESULT" |

16、内置引擎关键词提取

智能标引中关键词抽取，首先是通过词典对文献进行切词，然后利用词典中所规定词条权重的不同，从文本中自动抽取能够高度有效表达文本主题和内容的词汇的过程。

语法格式:

NLPE EXTRACTWORD

SET language=<1|2>, withclass=<0|1>, closedict=<布尔值>

OutputSrcTable=<布尔值>

From Table = <表1[,表2,…]> with <(列1,权重)>[ <(列2,权重)>,…]>

[StartRec = <整数> EndRec = <整数> ]

To [Table = <表名>] Field = <字段名>

参数:

language为处理数据的语言类型，1为中文；2为英文。

Withclass为返回结果是否带类别，0为不带；1为带。

Closedict指定是否执行完操作后卸载词典资源。

举例:

将库表cjfd2005使用内置引擎提取关键词，生成的结果保存在源表字段word\_result中。

|  |  |
| --- | --- |
| **use ext KExt\_NLPE** |  |
| **NLPE EXTRACTWORD** |  |
| **SET** | language=2, withclass=0, closedict=false |
| **From TABLE =** | CJFD2005 WITH (title, 篇名)(abs, 中文摘要)(keyword, 中文关键词)(km, 中文刊名)(content, 全文)(oldclass, 分类号) |
|  | startrec = 1, endrec = -1 |
| **ON** | CJFD |
| **TO FIELD =** | word\_result |

17、内置引擎文本分类

内置引擎分类器是以选定表字段为依据，对表记录间接按照中图分类号进行分类。

语法格式:

NLPE EZTC

SET language=<1|2>, filter=<布尔值>, classtype==<1|2>, mode=<0|1>, closedict=<布尔值>

OutputSrcTable=<布尔值>

From Table = <表1[,表2,…]> with <(列1,权重)>[ <(列2,权重)>,…]>

[StartRec = <整数> EndRec = <整数> ]

ON DICT=""

To [Table = <表名>] Field = <字段名>

参数:

language为处理数据的语言类型，1为中文；2为英文。

Filter指定输出结果是否需要经过域值过滤。

classtype为分类数据的期刊类型，1为期刊库数据表；2为报纸库数据表。

mode为指定输出结果中是否包含中图分类号。

Closedict指定是否执行完操作后卸载词典资源。

举例:

将库表cjfd2005使用内置引擎自动文本分类，结果经过域值过滤，输出中图分类号，处理表前100条记录，生成的结果保存在表NLPE\_RESULT 的 subject\_result字段中。

|  |  |
| --- | --- |
| **use ext KExt\_NLPE** |  |
| **NLPE EZTC** |  |
| **SET** | language=1, filter=false, classtype=1, mode=1, closedict=false |
| **From TABLE =** | CJFD2005 WITH (title, 篇名)(abs, 中文摘要)(keyword, 中文关键词)(km, 中文刊名)(content, 全文) |
|  | startrec = 1, endrec = -1 |
| **ON DICT =** | "" |
| **TO** | TABLE = NLPE\_RESULT FIELD = subject\_result |

附 录

附录一： KSQL的分词规则

KBase数据库目前的分词规则取决于其自然语言处理引擎和概念关系词典。目前，词条的规模达到500万。在这些词条中，大部分是专业术语和词汇，覆盖了中国图书分类体系中的所有类型和专业。随着专业的进一步细分，词典的规模还会继续增大。

1、切词器

实现全切分切词算法，利用语法信息、概率信息等，指导切词过程，从多个切词方案中选择最优的一个。例如，句子“论文化产业”，共有两个候选的切词方案，“论/文化/产业”和“论文/化/产业”，我们利用每个方案中各个词的语法信息和概率信息，计算各方案的切词概率，选择概率最大的一个方案，从而得到正确的切词结果。

实现正逆向最大长度匹配、最小长度匹配等切词算法。

切词速度达到世界领先水平，全切分切词算法的平均速度为2400万字/分钟，正向最大匹配算法的平均速度为1.7亿字/分钟。

切词准确率达到世界领先水平，全切分切词算法在封闭测试中可达到99%以上，在开放测试中可达到98%以上。

实现对超大规模词典的支持，在保证切词速度的情况下，尽量降低对内存的要求。自然语言处理引擎的切词器可以支持超大规模的词典，例如含500万词条的词典，所占内存在可接受的范围内。

尽量解决歧义切分问题，全切分算法可解决80%以上的切分歧义。

2、命名实体抽取器

实现人名、地名、机构名、专有名词等命名实体的辨识和自动提取，准确率在80%以上，可满足现有中文系统文本处理的需要。

目前，自然语言处理还在不断的研发新的功能，包括自动纠错、新词（术语、缩略语）的快速发现、面向自然语言的查询等。

附录二： KSQL的保留字

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** |  |  |  |  |
|  | ADD | ALIASNAME | ALL | ALTER |
|  | AND | AS | ASC | AT |
|  | AUTO | AVG |  |  |
| **B** |  |  |  |  |
|  | BY |  |  |  |
| **C** |  |  |  |  |
|  | CREATE | CREATEVIEW | COUNT |  |
| **D** |  |  |  |  |
|  | DATABASE | DATE | DEC | DEFAULT |
|  | DELETE | DESC | DISPLAYNAME | DISTINCT |
|  | DROP | DUPDB |  |  |
| **F** |  |  |  |  |
|  | FACTOR | FROM |  |  |
| **G** |  |  |  |  |
|  | GROUP |  |  |  |
| **I** |  |  |  |  |
|  | INDEX | INSERT | INTO | IS |
| **L** |  |  |  |  |
|  | LIKE | LOWER | LTRIM |  |
| **M** |  |  |  |  |
|  | MAINDB | MANUAL | MAX | MIN |
| **N** |  |  |  |  |
|  | NOT | NULL |  |  |
| **O** |  |  |  |  |
|  | ON | OR | ORDER |  |
| **P** |  |  |  |  |
|  | PACK | PATH |  |  |
| **R** |  |  |  |  |
|  | REFCOL | RELEVANT | REPLACE | RTRIM |
| **S** |  |  |  |  |
|  | SELECT | SET | SUBSTR | SUM |
| **T** |  |  |  |  |
|  | TABLE | TRIM |  |  |
| **U** |  |  |  |  |
|  | UPDATE | UPPER | USING |  |
| **V** |  |  |  |  |
|  | VALUES | VARSELECT | VIEW |  |
| **W** |  |  |  |  |
|  | WHERE | WITH |  |  |
| **X** |  |  |  |  |
|  | XLS |  |  |  |

附录三： 汉字编码 ＧＢＫ

GBK：汉字国标扩展码,基本上采用了原来GB2312-80所有的汉字及码位，并涵盖了原Unicode中所有的汉字20902，总共收录了883个符号， 21003个汉字及提供了1894个造字码位。 Microsoft简体版中文Windows 95就是以GBK为内码，又由于GBK同时也涵盖了Unicode所有CJK汉字，所以也可以和Unicode做一一对应。

GB码：全称是GB2312-80《信息交换用汉字编码字符集 基本集》，1980年发布，是中文信息处理的国家标准，在大陆及海外使用简体中文的地区（如新加坡等）是强制使用的唯一中文编码。P-Windows3.2和苹果OS就是以GB2312为基本汉字编码， Windows 95/98则以GBK为基本汉字编码、但兼容支持GB2312。GB码共收录6763个简体汉字、682个符号，其中汉字部分：一级字3755，以拼音排序，二级字3008，以偏旁排序。该标准的制定和应用为规范、推动中文信息化进程起了很大作用。

GBK编码是中国大陆制订的、等同于UCS的新的中文编码扩展国家标准。GBK工作小组于1995年10月，同年12月完成GBK规范。该编码标准兼容GB2312，共收录汉字21003个、符号883个，并提供1894个造字码位，简、繁体字融于一库。

附录四：样本数据库