ALIBABA CLOUD

阿里云

专有云企业版

表格存储Tablestore

用户指南

产品版本：v3.16.2

文档版本：20220915

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

3. 由于产品版本升级、调整或其他原因，本文档内容有可能变更。阿里云保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在阿里云授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

格式	说明	样例
危险	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	危险 重置操作将丢失用户配置数据。
警告	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	警告 重启操作将导致业务中断，恢复业务时间约十分钟。
注意	用于警示信息、补充说明等，是用户必须了解的内容。	注意 权重设置为0，该服务器不会再接受新请求。
说明	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	说明 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击**设置**> **网络**> **设置网络类型**。
粗体	表示按键、菜单、页面名称等UI元素。	在**结果确认**页面，单击**确定**。
Courier字体	命令或代码。	执行 `cd /d C:/window` 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	`bae log list --instanceid` *Instance_ID*
[] 或者 [a|b]	表示可选项，至多选择一个。	`ipconfig [-all|-t]`
{} 或者 {a|b}	表示必选项，至多选择一个。	`switch {active|stand}`

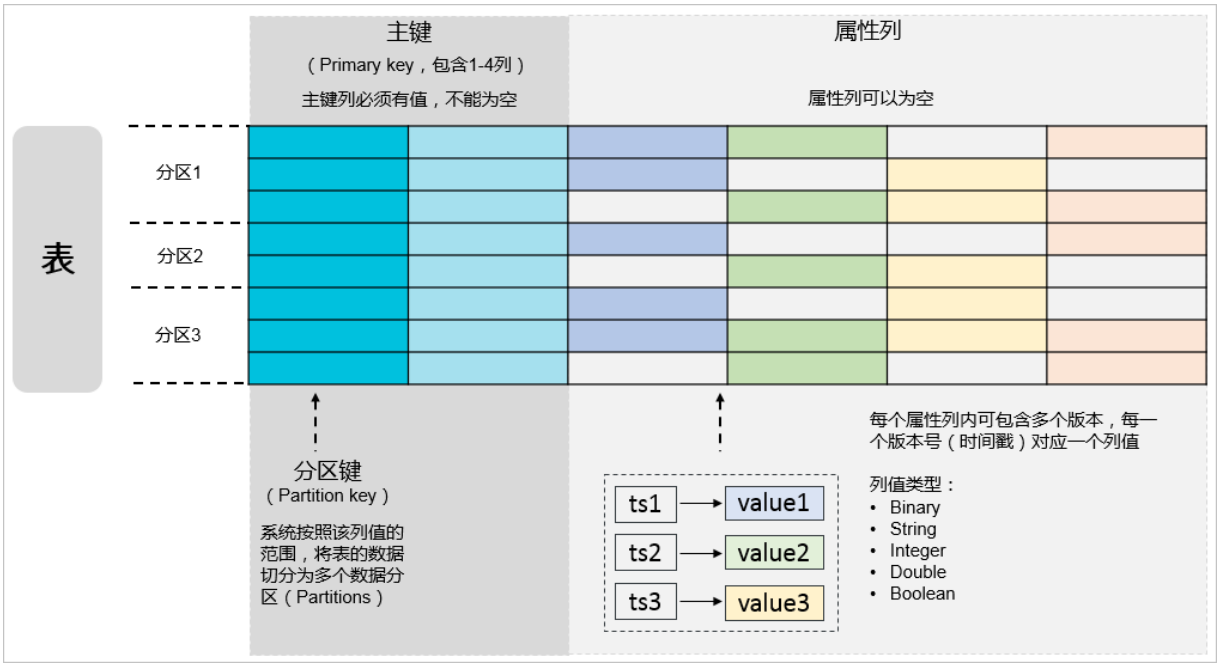
目录

1.基本概念

介绍产品的基本概念，包括数据模型、最大版本数、数据生命周期、有效版本偏差、主键和属性、读/写吞吐量、地域、实例、服务地址和SATA。

数据模型

表格存储的数据模型概念包括表、行、主键和属性，如下图所示。



最大版本数

最大版本数（Max Versions）是数据表的一个属性，表示该数据表中的属性列能够保留多少个版本的数据。当一个属性列的版本个数超过Max Versions时，最早的版本将被异步删除。

数据生命周期

数据生命周期TTL（Time To Live）是数据表的一个属性，即数据的存活时间，单位为秒。表格存储会在后台对超过存活时间的数据进行清理，以减少用户的数据存储空间，降低存储成本。

有效版本偏差

有效版本偏差（Max Version Offset）是数据表的一个属性，单位为秒。

为了防止非期望的写入，服务端在处理写请求时会对属性列的版本号进行检查。当版本号小于当前写入时间减去Max Version Offset，或者大于等于当前写入时间加上Max Version Offset的值时，该行数据写入失败。

属性列的有效版本范围为**[max{数据写入时间-有效版本偏差,数据写入时间-数据生命周期}，数据写入时间+有效版本偏差)**。数据写入时间为1970-01-01 00:00:00 UTC时间到当前写入时间的秒数。属性列版本号为毫秒，其除以1000换算成秒之后必须属于这个范围。

主键和属性

主键是表中每一行的唯一标识。主键由1到4个主键列组成。创建表的时候，必须明确指定主键的组成、每一个主键列的名字和数据类型以及它们的顺序。主键列的数据类型只能是String、Integer和Binary。如果为String或者Binary类型，长度不超过1 KB。

属性是属性存放行的数据。每一行包含的属性列个数没有限制。

读/写吞吐量

读/写吞吐量的单位为读服务能力单元和写服务能力单元，简称CU（Capacity Unit）。

地域

地域（Region）是指物理的数据中心，表格存储服务会部署在多个阿里云地域中，用户可以根据自身的业务需求选择不同地域的表格存储服务。

实例

实例（Instance）是用户使用和管理表格存储服务的实体，对应于关系型数据库的Database。实例是表格存储资源管理的基础单元，表格存储对应用程序的访问控制和资源计量都在实例级别完成。

服务地址

每个实例对应一个服务地址（EndPoint），应用程序在进行表和数据操作时需要指定服务地址。

SATA

SATA（Serial ATA）口的硬盘又叫串口硬盘。Serial ATA采用串行连接方式，具备了更强的纠错能力，旨在提高数据传输的可靠性。

2.使用限制

在使用表格存储前，您需要了解使用过程中的一些注意事项或者使用限制等。

产品的使用限制汇总请参见下表，部分限制范围表示用户能够使用的最大值，而不是建议值。为保证更好的性能，请合理设计表结构和单行数据大小。

限制项	限制范围	说明
一个阿里云账号下可以保有实例数	1024	如有需求提高上限，请联系管理员。
一个实例中表的个数	1024	如有需求提高上限，请联系管理员。
实例名称长度	3 ~ 16 bytes	实例名称需由a~z、A~Z、0~9和短划线（-）组成，首字符必须是字母且末尾字符不能为短划线（-）。
表名长度	1 ~ 255 bytes	表名需由a~z、A~Z、0~9和下划线（_）组成。首字符必须是字母或下划线（_）。
列名长度限制	1 ~ 255 bytes	列名需由a~z、A~Z、0~9和下划线（_）组成。首字符必须是字母或下划线（_）。
主键包含的列数	1~4	至少1列，至多4列。
String类型主键列的列值大小	1 KB	单一主键列String类型列的列值大小上限1 KB。
String类型属性列的列值大小	2 MB	单一属性列String类型列的列值大小上限2 MB。
Binary类型主键列的列值大小	1 KB	单一主键列Binary类型列的列值大小上限1 KB。
Binary类型属性列的列值大小	2 MB	单一属性列Binary类型列的列值大小上限2 MB。
一行中属性列的个数	不限制	不限制单一行拥有的属性列个数。
一次请求写入的属性列的个数	1024列	PutRow、UpdateRow或BatchWriteRow操作时，单行写入的属性列的个数不能超过1024列。
单行数据大小	不限制	不限制单一行中所有列名与列值总和大小。
读请求中columns_to_get参数的列的个数	0~128	读请求一行数据中获取的列的最大个数。
单表UpdateTable的次数	上调：无限制 下调：无限制	需要遵循单表的调整频率限制。
单表UpdateTable的频率	每2分钟1次	单表在2分钟之内，最多允许调整1次预留读/写能力值。
BatchGetRow一次操作请求读取的行数	100	无
BatchWriteRow一次操作请求写入行数	200	无
BatchWriteRow一次操作的数据大小	4 MB	无

限制项	限制范围	说明
GetRange一次返回的数据	5000行或者4 MB	一次返回数据的行数超过5000行，或者返回数据的数据大小大于4 MB。以上任一条件满足时，超出上限的数据将会按行级别被截掉并返回下一行数据主键信息。
一次HTTP请求Request Body的数据大小	5 MB	无

3.快速入门

3.1. 登录表格存储管理控制台

本文介绍如何登录表格存储管理控制台。

前提条件

- 登录Apsara Uni-manager运营控制台前，确认您已从部署人员处获取Apsara Uni-manager运营控制台的服务域名地址。
- 推荐使用Chrome浏览器。

操作步骤

1. 在浏览器地址栏中，输入Apsara Uni-manager运营控制台的服务域名地址，按回车键。
2. 输入正确的用户名及密码。

   请向运营管理员获取登录控制台的用户名和密码。

   > **说明** 首次登录Apsara Uni-manager运营控制台时，需要修改登录用户名的密码，请按照提示完成密码修改。为提高安全性，密码长度必须为10~32位，且至少包含以下两种类型：
   > - 英文大写或小写字母（A~Z、a~z）
   > - 阿拉伯数字（0~9）
   > - 特殊符号（感叹号（!）、at（@）、井号（#）、美元符号（$）、百分号（%）等）

3. 单击**账号登录**。
4. 如果账号已激活MFA多因素认证，请根据以下两种情况进行操作：
   - 管理员强制开启MFA后的首次登录：
     a. 在绑定虚拟MFA设备页面中，按页面提示步骤绑定MFA设备。
     b. 按照步骤2重新输入账号和密码，单击**账号登录**。
     c. 输入6位MFA码后单击**认证**。
   - 您已开启并绑定MFA：

     输入6位MFA码后单击**认证**。

     > **说明** 绑定并开启MFA的操作请参见*Apsara Uni-manager运营控制台用户指南*中的*绑定并开启虚拟MFA设备*章节。

5. 在顶部菜单栏中，选择**产品 > 存储 > 表格存储Tablestore**。

3.2. 创建实例

表格存储实例是用户使用和管理表格存储服务的实体，实例是表格存储资源管理的基础单元，表格存储对应用程序的访问控制和资源计量都在实例级别完成。本文主要介绍如何创建实例。

操作步骤

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击**创建实例**。

   > **说明** 您可以为不同业务创建不同的实例来管理相关的表，也可以为同一个业务的开发测试和生产环境创建不同的实例。默认情况下，表格存储允许一个云账号最多创建1024个实例，每个实例内最多创建1024张表。

3. 在**创建表格存储Tablestore实例**页面，按照以下说明进行配置。

配置项	说明
地域	选择实例所在的地域。
组织	选择实例所属的组织。
资源集	选择实例所属的资源集。
实例名称	输入实例的名称。 实例名称命名规范：3~16个字符，以大小写字母开头，不能以任意大小写的 `ali` 或 `ots` 开头，只能包含字母、数字和短划线（-）。
描述	实例的描述信息。
实例规格	实例的实例规格。实例规格分为高性能实例和容量型实例，具体视您定制的集群情况而定。

4. 单击**提交**。
5. 在**提交成功**对话框，单击**返回管理控制台**。

   在**概览**页面，可查看已创建的实例。

   创建实例后，可以对实例执行如下操作。

   - 单击实例名称或者在实例操作列单击**实例管理**，然后在**实例管理**页面，单击不同页签执行不同操作。
     - 在**实例详情**页签，可以查看实例访问地址和实例基础信息，创建数据表以及操作管理数据表。
     - 在**实例监控**页签，可以按照不同时间不同指标分组或操作监控相应指标。
     - 在**网络管理**页签，可以绑定或解绑VPC，查看VPC实例列表等。
   - 在实例**操作**列单击**释放**，可以释放实例。

     > **注意**
     > - 释放实例前，请确保已删除实例中的所有数据表和已解除实例中绑定的VPC。
     > - 为避免冲突，在释放过程中，请不要创建与释放实例名称相同的实例。

3.3. 创建数据表

本文介绍如何在表格存储管理控制台创建数据表。

操作步骤

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击**创建数据表**。

   > **说明** 单个实例最多可以创建1024张数据表。

4. 在**创建数据表**对话框，配置**数据表名称**和**表主键**。

   配置项说明如下。

配置项	说明
数据表名称	数据表的名称，用于在实例中唯一标识数据表。 数据表的命名规则为由大小写字母、数字或下划线（_）组成，且只能以字母或下划线（_）开头，长度在1个~255个字节之间。
表主键	表中的一个或多个主键列，用于在数据表中唯一标识记录。 输入表主键名称并选择数据类型，单击**添加表主键**，可加入新的主键。 单表最多可设置4个主键，第一个主键默认为分区键。主键的配置及顺序设置后不能修改。 **说明** ○ 表格存储支持将主键列设置为自增列，每张表只能设置一个主键列为自增列，且分区键不能设置为自增列。 ○ 主键列设置为自增列后，在写入一行数据时，该主键列无需填值，表格存储会自动生成该主键列的值。自动生成的主键列的值在分区键内严格递增且唯一。 ○ 表主键名称的命名规则为由大小写字母、数字或下划线（_）组成，且只能以字母或下划线（_）开头，长度在1个~255个字节之间。 ○ 主键的数据类型可选为**字符串**、**整型**、**二进制**或**自增列**。 只有当主键为非分区键时，主键的数据类型才能选择为**自增列**。

5. （可选）配置高级参数。

   当需要配置数据生命周期、最大版本数等参数时，请执行此操作。

   i. 打开**高级设置**开关。

ii. 配置高级配置项。

配置项说明如下。

配置项	说明
数据生命周期	数据最长的存放时间，当数据的存放时间超过数据生命周期时，系统会自动删除过期数据。 数据生命周期至少为86400秒（一天）或为-1（数据永不过期）。
最大版本数	属性列数据最多能保留的版本个数，当属性列数据的版本个数超过最大版本数时，系统会自动删除较早版本的数据，直到属性列数据的版本个数等于最大版本数。 最大版本数取值范围为1~10且必须为非0值。
数据有效版本偏差	只有写入数据所有列的版本号与写入时间的差值在数据有效版本偏差范围内，数据才能成功写入。 属性列的有效版本范围为**[max{数据写入时间-有效版本偏差,数据写入时间-数据生命周期}，数据写入时间+有效版本偏差)**。
预留读吞吐量	只有高性能实例支持配置此参数。 为数据表分配和预留的读/写吞吐量。 取值范围为0~5000且取值必须为整数。 当预留读/写吞吐量配置为0时，表示不为数据表预留读/写吞吐量。
预留写吞吐量	

6. （可选）配置全局二级索引。

当需要使用全局二级索引时，请执行此操作。

i. 打开**创建二级索引**开关。

ii. 单击**添加预定义列**，输入预定义列的名称，并选择数据类型。

- 此处是为主表添加预定义列。表格存储为Schema-free模型，原则上一行数据可以写入任意列，无需在schema中指定。但也可以在创建数据表时预先定义一些列以及对应类型。
- 最多可以添加20个预定义列。如需删除已添加的预定义列，在对应预定义列的右侧单击 图标。
- 预定义列名称由大小写字母、数字或下划线（_）组成，且只能以字母或下划线（_）开头，长度在1个~255个字节之间。
- 预定义列的数据类型包括字符串、整型、二进制、浮点数、布尔值。

iii. 单击**添加二级索引**，输入索引名，选择索引主键、索引预定义列和**索引类型**。

- 索引名名称由大小写字母、数字或下划线（_）组成，且只能以字母或下划线（_）开头，长度在1个~255个字节之间。
- 索引主键可以是主表主键和预定义列。
- 索引预定义列只能从数据表的预定义列中选择。
- **索引类型**的取值范围为**全局**和**本地**，请根据实际选择。

7. 单击**确定**。

   数据表创建完成后，在**数据表列表**区域，可以查看已创建的数据表。如果新建的表未显示在列表中，请单击 图标，刷新数据表列表。

   数据表创建完成后，支持对数据表进行如下操作。

   - 单击表名或在操作列选择不同功能，然后在**表管理**页面，单击不同页签执行不同操作。
     - 在**基本详情**页签，可以查看数据表的基本信息、高级功能、主键列表、预定义列、Stream信息等。
     - 在**数据管理**页签，可以插入数据，更新数据，查询数据，查看数据详情，批量删除数据等。
     - 在**索引管理**页签，可以创建二级索引，查看索引详情，查询数据，删除索引等。
     - 在**实时消费通道**页签，可以开启Stream功能，创建通道，展示通道分区列表，删除通道等。
     - 在**监控指标**页签，可以查看表或者索引在指定时间范围内不同指标分组或操作类型的监控指标数据。监控指标包括服务监控总览、平均访问延迟、每秒请求次数、行数统计、流量统计等。
   - 在操作列单击 图标后选择**删除**，可以删除数据表。

> **注意** 删除数据表时，系统会永久删除数据表及表中数据，且删除后的数据表及表中数据无法恢复，请谨慎操作。

3.4. 控制台读写数据

数据表创建成功后，您可以在表格存储控制台进行数据读写。

写入数据

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称后选择**数据管理**页签或在操作列单击**查询/搜索**。
4. 在**数据管理**页签，单击**插入数据**。
5. 在**插入数据**对话框，输入主键的值，单击**增加属性列**，输入**属性列名称**、**属性列类型**、**属性值**和**数据版本号**。

   系统默认已选中**使用系统时间**复选框，采用当前系统时间作为数据版本号。您也可以取消**使用系统时间**复选框后，输入数据版本号。
6. 单击**确定**。

   写入的数据行会显示在**数据管理**页签。

更新数据

更新一行数据的属性列。

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称后选择**数据管理**页签或在操作列单击**查询/搜**

索。

4. 在**数据管理**页签，选中需要更新的数据行，单击**更新数据**。
5. 在**更新数据**对话框，增加或删除属性列，更新或删除属性列的数据。
   - 单击**增加属性列**，增加属性列；单击图标，删除属性列。
   - 选择更新操作为**更新**，修改属性列的数据；选择更新操作为**删除**，选择需要删除的数据版本号，删除对应数据版本号的数据；选择更新操作为**删除全部**，删除全部数据版本号的数据。
6. 单击**确定**。

读取数据

在控制台对数据表进行单行查询和范围查询。

单行查询的操作步骤如下：

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称后选择**数据管理**页签或在操作列单击**查询/搜索**。
4. 在**数据管理**页签，单击**查询数据**。
5. 设置查询条件。
   i. 在**查询数据**对话框，选择查询范围为**单行查询**。
   ii. 系统默认返回所有列，如需显示指定属性列，关闭**获取所有列**开关并输入需要返回的属性列，多个属性列之间用半角逗号（,）隔开。
   iii. 输入目标行的**主键值**。

   主键值的完整性和准确性均会影响查询。
   iv. 输入**最大版本数**，指定需要返回的版本数。

   > **说明** 控制台最多可返回20个版本的数据，使用SDK无限制。
6. 单击**确定**。

范围查询的操作步骤如下：

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称后选择**数据管理**页签或在操作列单击**查询/搜索**。
4. 在**数据管理**页签，单击**查询数据**。
5. 设置查询条件。
   i. 在**查询数据**对话框，选择查询范围为**范围查询**。
   ii. 系统默认返回所有列，如需显示指定属性列，关闭**获取所有列**开关并输入需要返回的属性列，多个属性列之间用半角逗号（,）隔开。

iii. 输入起始主键列和结束主键列。

起始主键列可选**最小值**或**自选**，结束主键列可选**最大值**或**自选**。当选择**自选**时，需要输入自定义值。

> **说明**
>
> - 范围查询优先使用第一个主键值进行查询，当设置的第一个主键值一致时，系统会使用第二个主键值进行查询，其他主键的查询规则同上。
> - 读取范围是前开后闭的区间。

iv. 输入**最大版本数**，指定需要返回的版本数。

> **说明** 控制台最多可返回20个版本的数据，使用SDK无限制。

v. 设置查询结果的排序方向，可选**正序查询**或**逆序查询**。

6. 单击**确定**。

删除数据

删除不需要的数据。

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称后选择**数据管理**页签或在操作列单击**查询/搜索**。
4. 在**数据管理**页签，选中需要删除的数据行，单击**批量删除**。
5. 在**批量删除**对话框，单击**确定**。

3.5. 绑定VPC

将VPC绑定到表格存储实例后，VPC内的ECS实例可以通过VPC网络访问同地域的表格存储实例。

前提条件

- 已创建和表格存储实例在相同地域的VPC。
- 专有网络创建成功后，已在VPC内创建ECS实例。

操作步骤

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 单击**网络管理**页签。
4. 在**网络管理**页签，单击**绑定VPC**。
5. 在**绑定VPC**对话框，选择目标VPC和交换机，输入**VPC Name**。

   VPC Name只能包含字母、数字，且必须以字母开头。名称长度在3~16个字节之间。
6. 单击**确定**。

绑定成功后，在**网络管理**页签的**VPC列表**中查看已绑定的VPC信息。该VPC中的ECS实例可以通过VPC地址访问绑定的表格存储实例。

绑定VPC信息后，可以执行如下操作。

- 在操作列单击**VPC实例列表**，可以查看VPC实例列表信息，包括实例名称、实例VPC名称、VPC域名等。
- 在操作列单击**解除绑定**，可以解除实例和VPC的绑定关系。解除绑定后，在该VPC内的ECS实例无法再通过VPC地址访问表格存储，如仍需访问请再次绑定。

3.6. 使用通道服务

开启Stream功能后，通过为数据表建立数据通道，您可以实现对表中历史存量和新增数据的消费处理。

背景信息

通道服务（Tunnel Service）是基于表格存储数据接口之上的全增量一体化服务。通道服务提供了增量、全量、全量加增量三种类型的分布式数据实时消费通道。

开启Stream功能

开启Stream功能后，当Stream操作日志存放时长超过设定的日志过期时长时，系统会定期删除过期的日志。

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称或者在操作列单击 ⋮ 图标后选择**实时消费通道**。
4. 在**实时消费通道**页签的**Stream信息**区域，单击**开启**。
5. 在**开启Stream功能**对话框，输入**日志过期时长**。

   > 说明
   > - **日志过期时长**是指Stream操作日志的过期时间。
   > - **日志过期时长**的单位是小时且值必须是非零整数，最长可设置为168小时，设置后不能修改。

6. 单击**开启**。

创建通道

为数据表建立数据通道，实现对表中历史存量和新增数据的消费处理。

1. 登录表格存储管理控制台。
2. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
3. 在**实例详情**页签的**数据表列表**区域，单击数据表名称或者在操作列单击 ⋮ 图标后选择**实时消费通道**。
4. 在**实时消费通道**页签，单击**创建通道**。
5. 在**创建通道**对话框，输入**通道名**，并选择**通道类型**。

   通道服务提供**增量**、**全量**、**全量加增量**三种类型的分布式数据实时消费通道。

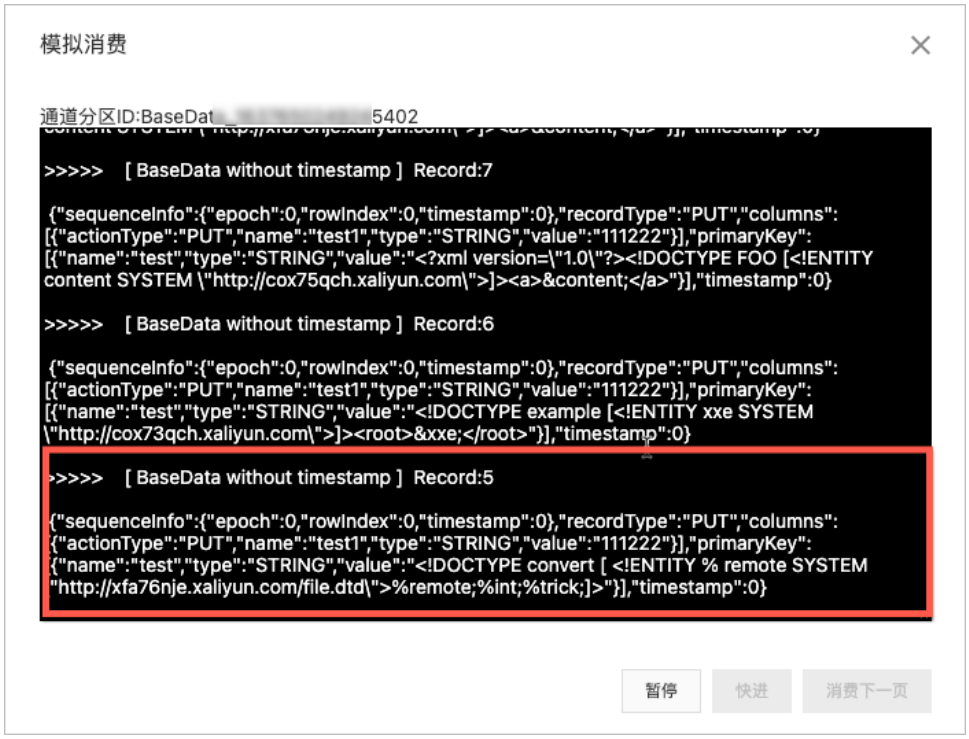
6. 单击**确定**。

   创建通道后，可以在**实时消费通道**页签中查看通道的信息。

预览通道中的数据格式

通过写入或删除数据，模拟数据消费，预览通道中的数据格式。

1. 写入或删除数据，详情请参见控制台读写数据。
2. 预览通道中的数据格式。
   i. 登录表格存储管理控制台。
   ii. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
   iii. 在**实例详情**页签的**数据表列表**区域，单击数据表名称或者在操作列单击 ⋮ 图标后选择**实时消费通道**。
   iv. 在**实时消费通道**页签，单击通道的**展示通道分区列表**。
   v. 在通道分区列表中，单击某一通道对应的**模拟消费**，并在弹出的对话框中单击**开始消费**。



开启通道的数据消费

获取已创建通道的ID，使用任一语言的通道SDK，开启通道的数据消费。

1. 获取已创建通道的ID。
   i. 登录表格存储管理控制台。
   ii. 在**概览**页面，单击实例名称或在操作列单击**实例管理**。
   iii. 在**实例详情**页签的**数据表列表**区域，单击数据表名称或者在操作列单击 ⋮ 图标后选择**实时消费通道**。
   iv. 在**实时消费通道**页签，复制已创建通道的ID。

2. 使用任一语言的通道SDK，开启通道的数据消费。

```
//用户自定义数据消费Callback，即实现IChannelProcessor接口（process和shutdown）。
private static class SimpleProcessor implements IChannelProcessor {
    @Override
    public void process(ProcessRecordsInput input) {
        System.out.println("Default record processor, would print records count");
        System.out.println(
            String.format("Process %d records, NextToken: %s", input.getRecords().size(
), input.getNextToken()));
        try {
            // Mock Record Process.
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
    @Override
    public void shutdown() {
        System.out.println("Mock shutdown");
    }
}
//在TunnelWorkerConfig中可以配置高级参数。
TunnelWorkerConfig config = new TunnelWorkerConfig(new SimpleProcessor());
//配置TunnelWorker，并启动自动化的数据处理任务。
TunnelWorker worker = new TunnelWorker($tunnelId, tunnelClient, config);
try {
    worker.connectAndWorking();
} catch (Exception e) {
    e.printStackTrace();
    worker.shutdown();
    tunnelClient.shutdown();
}
```

4.全局二级索引

4.1. 使用前须知

介绍全局二级索引的基本概念、使用限制以及注意事项。

基本概念

名词	描述
索引表	对主表某些列数据的索引，只能读不能写。
预定义列	表格存储为Schema-free模型，原则上一行数据可以写入任意列，无需在schema中指定。但是也可以在建表时预先定义一些列以及其类型。
单列索引	只为某一个列建立索引。
组合索引	多个列组合成索引，组合索引中包含组合索引列1、列2。
索引表属性列	被映射到索引表非PK列中的主表预定义列。
索引列补齐	自动将没有出现在用户指定索引列中的主表PK列补充到索引表PK中。

限制

- 同一实例下，索引表名称不允许重复。
- 同一张主表下，最多建立5张索引表。当超出索引表数目限制后，新建索引表将失败。
- 同一张主表下，最多建立15个预定义列。当超出预定义列数目限制后，新建主表将失败。
- 一张索引表的索引列最多有4列，索引列为主表PK和主表预定义列的任意组合。当超出限制后，新建索引表将失败。
- 一张索引表的属性列最多有8列。当超出限制后，新建索引表将失败。
- 索引列的类型为整型、字符串、二进制，与主表PK列的约束相同。
- 多个列组合成索引，大小限制与主表PK列相同。
- 类型为字符串或二进制的列，作为索引表的属性列时，限制与主表相同。
- 暂不支持TTL表建立索引，有需求请钉钉联系表格存储技术支持。
- 不支持在使用多版本功能的表上建立索引。如果表打开了多版本，建索引将失败；如果有索引，打开主表多版本功能将失败。
- 有索引的主表上，写入数据时，不允许自定义版本，否则主表写入数据将失败。
- 索引表上不允许使用Stream功能。
- 有索引表的主表上，同一个Batch写中，同一行（主键相同）不允许重复存在，否则主表写入数据将失败。

注意事项

- 对于每张索引表，系统会自动进行索引列补齐。在对索引表进行扫描时，您需要注意填充对应PK列的范围（一般为负无穷到正无穷）。例如：主表有PK0、PK1两列PK，Defined0一列预定义列。

如果您指定在Defined0列上建立索引，则系统会将索引表的PK生成为Defined0、PK0、PK1。您可以指定在Defined0列及PK1列上建立索引，则生成索引表的PK为Defined0、PK1、PK0。您还可以在PK1列上建立索引，则生成索引表的PK为PK1、PK0。您在建立索引表时，只需要指定您需要的索引列，其它列会由系统自动添加。例如主表有PK0、PK1两列PK，Defined0作为预定义列：

  - 如果在Defined0上建立索引，那么生成的索引表PK将会是Defined0、PK0、PK1三列。
  - 如果在PK1上建立索引，那么生成的索引表PK将会是PK1、PK0两列。

- 选择主表的哪些预定义列作为主表的属性列。将主表的一列预定义列作为索引表的属性列后，查询时不用反查主表即可得到该列的值，但是同时增加了相应的存储成本。反之则需要根据索引表反查主表。您可以根据您的查询模式以及成本的考虑，作出相应的选择。
- 不建议把时间相关列作为索引表PK的第一列，这样可能导致索引表更新速度变慢。建议将时间列进行哈希，然后在哈希后的列上建立索引，如果有类似需求可以钉钉联系表格存储技术支持一同讨论表结构。
- 不建议取值范围非常小，甚至可枚举的列作为索引表PK的第一列。例如性别，这样将导致索引表水平扩展能力受限，从而影响索引表写入性能。

4.2. 使用场景

全局二级索引支持在指定列上建立索引，生成的索引表中数据按您指定的索引列进行排序，主表的每一个写入都将以异步方式自动同步到索引表。您只向主表中写入数据，根据索引表进行查询，在许多场景下，将极大的提高查询的效率。本文为您介绍电话话单查询场景下如何使用全局二级索引。

以我们常见的电话话单查询为例，建立主表如下：

CellNumber	StartTime（Unix时间戳）	CalledNumber	Duration	BaseStationNumber
123456	1532574644	654321	60	1
234567	1532574714	765432	10	1
234567	1532574734	123456	20	3
345678	1532574795	123456	5	2
345678	1532574861	123456	100	2
456789	1532584054	345678	200	3

- `CellNumber`、`StartTime`作为表的联合主键，分别代表**主叫号码**与**通话发生时间**。
- `CalledNumber`、`Duration`、`BaseStationNumber`三列为表的预定义列，分别代表**被叫号码**、**通话时长**、**基站号码**。

每次用户通话结束后，都会将此次通话的信息记录到该表中。可以分别在**被叫号码**，**基站号码**列上建立二级索引，来满足不同角度的查询需求。

假设有以下几种查询需求：

- 查询号码`234567`的所有主叫话单。

  由于表格存储为全局有序模型，所有行按主键进行排序，并且提供顺序扫描（`getRange`）接口，所以只需要在调用`getRange`接口时，将PK0列的最大及最小值均设置为`234567`，PK1列（通话发生时间）的最小值设置为`0`，最大值设置为`INT_MAX`，对主表进行扫描即可：

```
private static void getRangeFromMainTable(SyncClient client, long cellNumber)
{
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(TABLE_NAME);
    //构造主键。
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.fromLo
ng(cellNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLo
ng(0));
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    //构造主键。
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.fromLong
(cellNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    rangeRowQueryCriteria.setMaxVersions(1);
    String strNum = String.format("%d", cellNumber);
    System.out.println("号码" + strNum + "的所有主叫话单:");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            System.out.println(row);
        }
        //若nextStartPrimaryKey不为null，则继续读取。
        if (getRangeResponse.getNextStartPrimaryKey() != null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

- 查询号码 `123456` 的被叫话单。

  表格存储的模型是对所有行按照主键进行排序，由于被叫号码存在于表的预定义列中，所以无法进行快速查询。因此可以在 `被叫号码` 索引表上进行查询。

  索引表 `IndexOnBeCalledNumber` ：

PK0	PK1	PK2
CalledNumber	CellNumber	StartTime
123456	234567	1532574734
123456	345678	1532574795
123456	345678	1532574861

PK0	PK1	PK2
654321	123456	1532574644
765432	234567	1532574714
345678	456789	1532584054

> **说明** 系统会自动进行索引列补齐。即把主表的PK添加到索引列后面，共同作为索引表的PK。所以索引表中有三列PK。

由于索引表 `IndexOnBeCalledNumber` 是按被叫号码作为主键，可以直接扫描索引表得到结果：

```
private static void getRangeFromIndexTable(SyncClient client, long cellNumber) {
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(INDEX0_NAME);
    //构造主键。
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_1, PrimaryKeyValue.fromLo
ng(cellNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MI
N);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.INF_MI
N);
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    //构造主键。
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_1, PrimaryKeyValue.fromLong
(cellNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MAX)
;
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    rangeRowQueryCriteria.setMaxVersions(1);
    String strNum = String.format("%d", cellNumber);
    System.out.println("号码" + strNum + "的所有被叫话单:");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            System.out.println(row);
        }
        //若nextStartPrimaryKey不为null，则继续读取。
        if (getRangeResponse.getNextStartPrimaryKey() != null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

- 查询基站 `002` 从时间 `1532574740` 开始的所有话单。

  与上述示例类似，但是查询不仅把 **基站号码** 列作为条件，同时把 **通话发生时间** 列作为查询条件，因此我们可以在 **基站号码** 和 **通话发生时间** 列上建立组合索引。

  索引表 `IndexOnBaseStation1` ：

PK0	PK1	PK2
BaseStationNumber	StartTime	CellNumber
001	1532574644	123456
001	1532574714	234567
002	1532574795	345678
002	1532574861	345678
003	1532574734	234567
003	1532584054	456789

  然后在 `IndexOnBaseStation1` 索引表上进行查询：

```
private static void getRangeFromIndexTable(SyncClient client,
                                           long baseStationNumber,
                                           long startTime) {
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(INDEX1_NAME);
    //构造主键。
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLo
ng(baseStationNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLo
ng(startTime));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MI
N);
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    //构造主键。
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLong
(baseStationNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.INF_MAX)
;
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    rangeRowQueryCriteria.setMaxVersions(1);
    String strBaseStationNum = String.format("%d", baseStationNumber);
    String strStartTime = String.format("%d", startTime);
    System.out.println("基站" + strBaseStationNum + "从时间" + strStartTime + "开始的所有被
叫话单:");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            System.out.println(row);
        }
        //若nextStartPrimaryKey不为null，则继续读取。
        if (getRangeResponse.getNextStartPrimaryKey() != null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

- 查询发生在基站 `003` 上时间从 `1532574861` 到 `1532584054` 的所有通话记录的通话时长。

  在该查询中不仅把 `基站号码` 列与 `通话发生时间` 列作为查询条件，而且只把 `通话时长` 列作为返回结果。您可以使用上一个查询中的索引，查索引表成功后反查主表得到通话时长：

```
private static void getRowFromIndexAndMainTable(SyncClient client,
                                                long baseStationNumber,
                                                long startTime,
                                                long endTime,
                                                String colName) {
```

```
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(INDEX1_NAME);
    //构造主键。
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLo
ng(baseStationNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLo
ng(startTime));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MI
N);
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    //构造主键。
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLong
(baseStationNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLong
(endTime));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    rangeRowQueryCriteria.setMaxVersions(1);
    String strBaseStationNum = String.format("%d", baseStationNumber);
    String strStartTime = String.format("%d", startTime);
    String strEndTime = String.format("%d", endTime);
    System.out.println("基站" + strBaseStationNum + "从时间" + strStartTime + "到" + strEnd
Time + "的所有话单通话时长:");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            PrimaryKey curIndexPrimaryKey = row.getPrimaryKey();
            PrimaryKeyColumn mainCalledNumber = curIndexPrimaryKey.getPrimaryKeyColumn(PR
IMARY_KEY_NAME_1);
            PrimaryKeyColumn callStartTime = curIndexPrimaryKey.getPrimaryKeyColumn(PRIMA
RY_KEY_NAME_2);
            PrimaryKeyBuilder mainTablePKBuilder = PrimaryKeyBuilder.createPrimaryKeyBuil
der();
            mainTablePKBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, mainCalledNumber.g
etValue());
            mainTablePKBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, callStartTime.getV
alue());
            PrimaryKey mainTablePK = mainTablePKBuilder.build(); // 构造主表PK
            //反查主表。
            SingleRowQueryCriteria criteria = new SingleRowQueryCriteria(TABLE_NAME, main
TablePK);
            criteria.addColumnsToGet(colName); //读取主表的"通话时长"列。
            //设置读取最新版本。
            criteria.setMaxVersions(1);
            GetRowResponse getRowResponse = client.getRow(new GetRowRequest(criteria));
            Row mainTableRow = getRowResponse.getRow();
            System.out.println(mainTableRow);
        }
        //若nextStartPrimaryKey不为null，则继续读取。
        if (getRangeResponse.getNextStartPrimaryKey() != null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
```

```
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

为了提高查询效率，可以在 `基站号码` 列与 `通话发生时间` 列上建立组合索引，并把 `通话时长` 列作为索引表的属性列：

数据库中的记录将会如下所示：

索引表 `IndexOnBaseStation2` ：

PK0	PK1	PK2	Defined0
BaseStationNumber	StartTime	CellNumber	Duration
001	1532574644	123456	60
001	1532574714	234567	10
002	1532574795	345678	5
002	1532574861	345678	100
003	1532574734	234567	20
003	1532584054	456789	200

然后在 `IndexOnBaseStation2` 索引表上进行查询：

```
private static void getRangeFromIndexTable(SyncClient client,
                                           long baseStationNumber,
                                           long startTime,
                                           long endTime,
                                           String colName) {
    RangeRowQueryCriteria rangeRowQueryCriteria = new RangeRowQueryCriteria(INDEX2_NAME);
    //构造主键。
    PrimaryKeyBuilder startPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder(
);
    startPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLo
ng(baseStationNumber));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLo
ng(startTime));
    startPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MI
N);
    rangeRowQueryCriteria.setInclusiveStartPrimaryKey(startPrimaryKeyBuilder.build());
    //构造主键。
    PrimaryKeyBuilder endPrimaryKeyBuilder = PrimaryKeyBuilder.createPrimaryKeyBuilder();
    endPrimaryKeyBuilder.addPrimaryKeyColumn(DEFINED_COL_NAME_3, PrimaryKeyValue.fromLong
(baseStationNumber));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_2, PrimaryKeyValue.fromLong
(endTime));
    endPrimaryKeyBuilder.addPrimaryKeyColumn(PRIMARY_KEY_NAME_1, PrimaryKeyValue.INF_MAX)
;
    rangeRowQueryCriteria.setExclusiveEndPrimaryKey(endPrimaryKeyBuilder.build());
    //设置读取列。
    rangeRowQueryCriteria.addColumnsToGet(colName);
    rangeRowQueryCriteria.setMaxVersions(1);
    String strBaseStationNum = String.format("%d", baseStationNumber);
    String strStartTime = String.format("%d", startTime);
    String strEndTime = String.format("%d", endTime);
    System.out.println("基站" + strBaseStationNum + "从时间" + strStartTime + "到" + strEnd
Time + "的所有话单通话时长:");
    while (true) {
        GetRangeResponse getRangeResponse = client.getRange(new GetRangeRequest(rangeRowQ
ueryCriteria));
        for (Row row : getRangeResponse.getRows()) {
            System.out.println(row);
        }
        //若nextStartPrimaryKey不为null，则继续读取。
        if (getRangeResponse.getNextStartPrimaryKey() != null) {
            rangeRowQueryCriteria.setInclusiveStartPrimaryKey(getRangeResponse.getNextSta
rtPrimaryKey());
        } else {
            break;
        }
    }
}
```

由此可见，如果不把 `通话时长` 列作为索引表的属性列，在每次查询时都需先从索引表中解出主表的PK，然后对主表进行随机读。当然，把 `通话时长` 列作为索引表的属性列后，该列被同时存储在了主表及索引表中，增加了总的存储空间占用。

- 查询发生在基站 `003` 上时间从 `1532574861` 到 `1532584054` 的所有通话记录的总通话时长，平均通话时长，最大通话时长，最小通话时长。

  相对于上一条查询，这里不要求返回每一条通话记录的时长，只要求返回所有通话时长的统计信息。用户可以使用与上条查询相同的的查询方式，然后自行对返回的每条通话时长做计算并得到最终结果。也可以使用SQL-on-OTS，省去客户端的计算，直接使用SQL语句返回最终统计结果。其兼容绝大多数MySql语法，可以更方便的进行更复杂的、更贴近用户业务逻辑的计算。