

SAP HANA™ 数据库 – 开发指南

– 如何使用SQL和SQLScript数据建模

1. 简介

1.1 什么是 SAP HANA

1.2 相关文档

2. SAP HANA 数据库概念

2.1 基本概念

2.1.1 现代化硬件对数据库系统架构的影响

2.1.2 基于列式和行式的存储

2.1.3 列式表的优势

2.2 架构概览

2.3 SAP HANA 数据库概念：表，模型和视图处理

2.3.1 表，试图和星型结构

2.3.2 SAP HANA 模型视图

2.3.3 SAP HANA 视图处理

3. 教程

3.1 使用 SAP HANA studio

3.2 如何使用 SAP HANA modeler

3.2.1 如何建立分析试图

3.2.2 查看表

3.2.3 建立分析试图

3.2.4 MANDT 属性

3.2.5 过滤器的应用

3.2.6 创建属性试图

3.2.7 创建简单的星型架构

3.2.8 计算属性

3.2.9 多语种属性视图（文本视图）

3.2.10 图形化计算视图

3.2.11 如何创建脚本计算视图

3.3 使用 SAP HANA studio 执行 SQL 和 SQLScript 语句

3.4 如何显示查询计划

3.4.1 查询计划中的列

3.4.2 查询计划中的 OPERATOR_NAME 列

3.5 使用 jdbc 驱动

3.5.1 安装驱动

3.5.2 系统必备

3.5.3 jdbc 驱动的整合

3.5.4 加载 jdbc 驱动

3.5.5 连接地址

3.5.6 jdbc 4.0 标准扩展 api

3.5.7 jdbc 跟踪记录

3.5.8 匹配 SQL 和 java 类型

4. 最佳实践

4.1 列式存储引擎的特点

4.2 SQL 查询代价的估算

4.2.1 行式搜索代价模型

4.2.2 列式搜索代价模型

4.3 SQL 查询引擎的优化技巧

4.3.1 表达式

4.3.2 联接

4.3.3 EXISTS / IN 运算符

4.3.4 set 操作

4.4 SQLScript 推荐实践

4.4.1 减少 SQL 语句的复杂度

4.4.2 识别共同的子表达式

4.4.3 多层聚集

4.4.4 理解语句执行的代价

4.4.5 充分利用底层引擎

4.4.6 减少依赖性

4.4.7 模拟 SQL 语句中的函数调用

4.4.8 避免混合使用计算引擎操作符和 SQL 查询

4.4.9 避免使用游标

4.4.10 避免使用动态 SQL

4.4.11 跟踪和调试

免责声明

本文档为预览版本，文档中的内容均不是来自 SAP AG 公司发布的任何关于 SAP 的产品，策略，或未来的发展的官方声明。文档中的信息可能是不完整或是不正确的。SAP 公司不假设，不保证，对本文档为您的商业用途或利用它产生的任何损失不承担责任。本文档应当在您已授权 SAP HANA 许可下使用，并受其条款指导您使用 SAP HANA。

1. 简介

1.1 什么是 SAP HANA?

SAP HANA 是由 SAP 带给您的一项令人兴奋的新技术。其核心利用了创新型的内存技术来存储数据，特别适合处理数据量非常大的表格型或关系型的数据，具有前所未有的性能。常见的数据库以行方式存储表格数据，例如，描述一个地址的所有数据都存储在内存中相互毗邻的位置。如果你的需求只是访问一个地址，程序会运行的很快，因为所有的数据是连续存储的。然而，试想这样的场景，你的程序需要计算有多少已储存的地址与特定的国家，城市或邮编对应？这种情况下，就可能不得不扫描整张表，选出每一行，然后检查国家或城市是否是需要的。由于所有的大容量存储设备，例如硬盘，以一种与感兴趣的数据相比很大的一整块形式访问数据，例如 512 字节的硬盘，很可能该设备读取一至多行的数据只是为了查找几个字符，比如“巴西”或是“旧金山”。业务数据表经常含有很多偶尔使用的数据字段或列，例如和其他表相关联的数据，或者控制其他字段使用的数据字段。你能想象如果你的程序可以绕开不必要的列而访问真正需要的信息所带来的效率提升吗？

如果使用这种数据存储方式，你将会体验到数据库或程序明显更快的响应。SAP HANA 通过高效的列式存储方式组织表来让你绕开读取不需要的数据。除了常见的行式存储架构之外，同样可以使用列式存储。这意味着你的程序无须等待数据库获取不需要的数据，因为列表中的所有数据都是以相邻方式储存。因此，在我们的地址表例子中，扫描列字段“国家”或“城市”比读取行式存储快很多。

但是，如果你的数据库系统已经把所有数据缓存到内存或是靠近处理器的快速读取内存中，列式内存布局是否仍然可以加速访问？来自 SAP 和位于波茨坦市的哈索-普拉特纳研究所进行的测量证明，当访问每一行数据的子集时，以列方式重新组织内存中的数据可以带来显著的速度提升。由于 SAP HANA 把数据都缓存至内存中，硬盘几乎很少使用，只是为了数据持久化而对数据库的改变进行记录。SAP HANA 为了保持数据库尽可能小的变化，采用只对原始数据库的增量变化记录的方式。数据是增加或插入到一个表列而不是就地修改，这种方式提供了很多的好处，不只是速度上的提升。由于保留了所有的旧数据，你的程序可以高效地在数据间“时空穿梭”，并提供随时间变化的数据的视图。

现代数据库系统把数据管理和数据应用分隔至两个独立的体系结构层：数据库层和数据应用层。这种分隔方式迫使数据在被分析或是修改前，不得不从数据库“漫游”到应用层，很多时候，数据量非常大。SAP HANA 通过下放数据密集的应用逻辑到数据本来的地方，即数据库本身，来避免这种常见的瓶颈。为了在数据库启用这种内置的应用逻辑，SAP 开发了标准 SQL 的扩展（结构化查询语言）名为 SQLScript。SQLScript 允许编程的方式使得在数据库层执行数据密集型业务，也允许你扩充 SQL 语句来包含高水平的计算，从而提升了数据库的数据处理能力。

本文档阐述了如何利用 SQLScript 在 SQL HANA 数据库中实现高效的密集型数据处理。

1.2 相关文档

还有其他相关文档，说明了程序员工具的细节和可以使用的编程语言，包含如下：

- [SAP HANA Database – Administration Guide](#)
- 如何使用 SAP HANA Studio 以及管理 SAP HANA 数据库.

[*SAP HANA Modeling Guide*](#)

- 如何利用HANA Modeler在SQLScript编程语言的基础上创建OLAP分析试图和计算试图。

[*SAP HANA Database – SQL Reference Guide \(PDF\)*](#)

[*SAP HANA Database – SQL Reference Guide \(HTML\)*](#)

- SAP HANA查询语言完整参考

[*SAP HANA Database – SQLScript Guide*](#)

- 有关如何在SAP HANA程序，包括ABAP程序中使用SQLScript和存储过程的教程。

2 SAP HANA 数据库概念

SAP HANA 数据库概念上来说是利用内存数据存储提升速度，增加数据库查询的执行速度，以及提高程序开发速度。查询在 SAP HANA 数据库中可以快速，并行的执行。这代表不再需要复杂的编程技巧，如提前计算值（物化聚集），用来维护传统数据库性能，因为你可以利用 HDB 实时地查询巨大的数据集。消除了开发的复杂性，程序可以更直接和清晰的方式创建，因此实现了更快的开发时间。

SAP HANA 也能很好胜任传统数据库的存储和访问，例如基于行式存储的表可供使用。这种传统和创新型技术的结合使得开发人员可以为程序选择最好的技术，并且在需要时二者可以并用。

2.1 基本概念

2.1.1 现代化硬件对数据库系统架构的影响

从历史角度来看，数据库系统曾被设计运行在有限内存的计算机系统，主要由于缓慢的磁盘输入输出是数据吞吐量的主要瓶颈。因此，这些系统结构的设计成关注于优化磁盘访问，例如通过最小化读入内存的磁盘块（或页）来处理查询语句。

近年来，计算机体系结构已经发生变化。现在多核处理器（多个处理器集中在一个芯片上或是一个包）已成为标准，伴随着因处理器内核之间的快速通信而实现的并行处理。主存已经不再是有限的资源，现代服务器能拥有达到 2TB 的系统内存，可以使整个数据库进驻在内存中。目前服务器处理器有多达 64 个内核，并且 128 个核心即将问世。随着内核数量的增加，处理器在每次时间间隔可以处理更多的数据。这样把性能瓶颈从磁盘输入输出转移到了处理器缓存和内存数据传输上。（见图 1）。

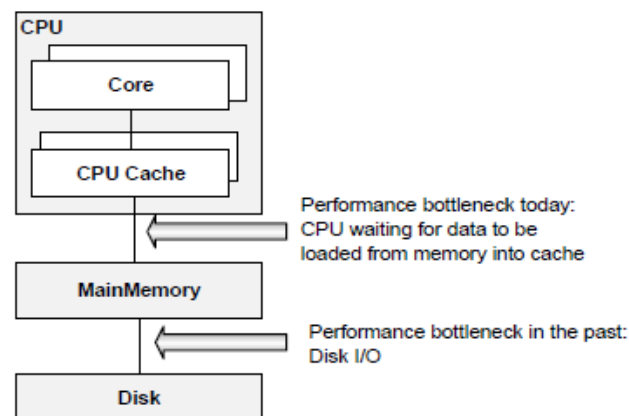


图 1 硬件体系：现在和过去的性能瓶颈

传统的数据库联机事务处理没有有效地利用当前的硬件。1999 年 Alamaki 等人证明，当数据库加载所有数据到内存时，处理器会停滞一半的执行时间，如等待数据从主存加载到处理器缓存中。

那么，运行在现代化硬件上的数据库系统的理想特点是什么？

内存数据库。所有有关的数据都在内存中。这一特点避免了硬盘读写的性能损失。由于所有的数据都在内存中，不再需要减少磁盘读写的磁盘索引。磁盘存储仍然要用来持久化，例如遇到断电事件。

缓存优化的内存结构。设计必须尽量减少一些缓存未命中以及避免因内存访问处理器造成的停滞。达到这个目的普遍机制是最大限度地提高数据的空间局部性，例如需要连续访问的数据应当存放在邻近的内存中。

支持并行执行。通过增加更多的内核到处理器中是时下实现更高的处理器执行速度的方法。早前，通过在芯片上应用更高的包装密度，优化电子式电流路径得到改善。利用这些技术取得的速度上的提升已经不明显了。多处理器需要新的并行算法，用来在数据库中充分利用现有的计算资源。

2.1.2 基于列式和行式的存储

正如上文提到，列式存储组织结构在特定场景下能减少内存的未命中，因此，更少的处理器停滞。这在处理器需要扫描整列时特别有用，比如执行查询时，索引搜索无法满足或者对于列进行的聚合运算，如求和或是求平均数。

索引行是加速访问基于行的表的传统方法，这在列式存储中也是行之有效的，但索引树的空间局部性很低，因此明显增加了高速缓存未命中。除此之外，每次数据插入表之后，索引必须重新组织。因此，数据库开发人员必须了解两个存储技术的优点和缺点，以便找到一个合适的平衡点。

从概念上讲，数据库表是一个二维的数据行和列组织的单元结构；而计算机内存是线性结构。为了在线性的内存中存储表，存在两个选择，如图 2 所示。基于行存储按照记录的顺序储存表，每条记录包含了行的所有列。相反，列式存储把一列的条目都放在连续的内存空间中。

Table	Row Store	Column Store																																														
<table><tr><th>Country</th><th>Product</th><th>Sales</th></tr><tr><td>US</td><td>Alpha</td><td>3.000</td></tr><tr><td>US</td><td>Beta</td><td>1.250</td></tr><tr><td>JP</td><td>Alpha</td><td>700</td></tr><tr><td>UK</td><td>Alpha</td><td>450</td></tr></table>	Country	Product	Sales	US	Alpha	3.000	US	Beta	1.250	JP	Alpha	700	UK	Alpha	450	<table><tr><td rowspan="3">Row 1</td><td>US</td></tr><tr><td>Alpha</td></tr><tr><td>3.000</td></tr><tr><td rowspan="3">Row 2</td><td>US</td></tr><tr><td>Beta</td></tr><tr><td>1.250</td></tr><tr><td rowspan="3">Row 3</td><td>JP</td></tr><tr><td>Alpha</td></tr><tr><td>700</td></tr><tr><td rowspan="3">Row 4</td><td>UK</td></tr><tr><td>Alpha</td></tr><tr><td>450</td></tr></table>	Row 1	US	Alpha	3.000	Row 2	US	Beta	1.250	Row 3	JP	Alpha	700	Row 4	UK	Alpha	450	<table><tr><td rowspan="4">Country</td><td>US</td></tr><tr><td>US</td></tr><tr><td>JP</td></tr><tr><td>UK</td></tr><tr><td rowspan="4">Product</td><td>Alpha</td></tr><tr><td>Beta</td></tr><tr><td>Alpha</td></tr><tr><td>Alpha</td></tr><tr><td rowspan="4">Sales</td><td>3.000</td></tr><tr><td>1.250</td></tr><tr><td>700</td></tr><tr><td>450</td></tr></table>	Country	US	US	JP	UK	Product	Alpha	Beta	Alpha	Alpha	Sales	3.000	1.250	700	450
Country	Product	Sales																																														
US	Alpha	3.000																																														
US	Beta	1.250																																														
JP	Alpha	700																																														
UK	Alpha	450																																														
Row 1	US																																															
	Alpha																																															
	3.000																																															
Row 2	US																																															
	Beta																																															
	1.250																																															
Row 3	JP																																															
	Alpha																																															
	700																																															
Row 4	UK																																															
	Alpha																																															
	450																																															
Country	US																																															
	US																																															
	JP																																															
	UK																																															
Product	Alpha																																															
	Beta																																															
	Alpha																																															
	Alpha																																															
Sales	3.000																																															
	1.250																																															
	700																																															
	450																																															

图 2 行式存储和列式存储

列式存储的概念已经使用了相当长的时间。从历史上看，它主要用于聚合功能发挥重要作用的分析和数据仓库。在联机事务处理中，使用列式存储需要以平衡的方式插入索引列的数据，以减少缓存的未命中。

SAP HANA 数据库可以让开发人员指定一张表是否以行存储或列存储，也可以修改已有的表结构从列式转换成行式，反之亦然。有关详细信息，请参阅 SAP HANA SQL 参考。

列式存储表的优势体现在下列情况中：

- 通常只对单列或几列进行计算
- 基于表中几列字段进行的搜索
- 表中有大量的列
- 表中有大量的行，并且需要进行列操作（聚合，扫描等）
- 大多数的列只包含几个不同的值，可以实现高压缩率

行式存储表的优势体现在下列情况中：

- 程序需要在同一时间内只处理单个记录（很多select和/或单个记录的更新）
- 程序通常需要访问整个记录（或行）
- 大多数列的值都不相同，压缩率很低
- 既不需要聚合也不需要快速搜索
- 表中只含有少数行（例如配置表）

为了利用快速动态聚合、特别报告，以及受益于压缩机制，建议将业务数据存储存储在列式表中。SAP HANA 能联接列式表和行式表，但是，相同存储方式表的联接更为高效。例如，需要经常和业务数据表联接的主数据表也应当采用列式存储。

2.1.3 列式表的优势

如果上文提到的准则都能满足，本节将解释列式存储表的优势。这不代表列式存储表总是最佳选择，还有很多情况下，行式存储有优势。

优势：更高的数据压缩率

利用压缩可以达到以更少的代价，保证所有相关数据都存放在内存中的目标。列式数据存储可以进行非常高效的数据压缩，特别是如果对列进行排序，一般都会有几个连续的值放在相邻的内存中。这种情况下，可以使用压缩方法，比如运行长度编码、集群编码或是字典编码，这对于业务程序是特别有用，因为表中的很多列包含与记录数相比数量很少的不同值。极端的例子，如国家代码或邮政编码的代码，其他有效的例子包括客户和帐户号码，地区代码，销售渠道的代码，或状态码。后者更多的被用作数据库中其他表的外键，例如表中含有客户订单和会计记录数据，对于这种高度冗余的列可以有效的压缩数据。在行式存储中，连续的内存空间存放了不同列的数据，因此，例如运行长度编码的压缩方法就无法使用。在列式存储中，相比传统的行式存储系统，压缩因子可以达到 5 倍。

压缩技术的简要讨论如下：

运行长度编码。如果对一列的数据进行排序，有两个或两个以上的元素包含相同的值的概率很高。运行长度编码方式是计算含有相同值的连续列的个数。为了实现这一目标，原有列将被新的两列取代。第一列存放的是原有列中的值而第二列存放的是各值出现的次数。有了这些信息可以很容易地重建原始列。

集群编码。这种压缩技术的工作原理是寻找原始列的值以相同顺序多次出现的次数。压缩后的列由两列组成，第一列存放的是指定顺序的元素，第二列存放原始列内该顺序起始行号。很多常用的数据压缩程序利用这一技术来压缩文本文件。

字典编码。对于列含有相对较少的不同值的表，可以通过枚举不同值和只存放它们的数字进行有效的压缩。这种技术需要额外的一张表，即字典表，第一列维护的是原始值而第二列则是代表原始值的数字。这种技术的压缩率很高，例如在国家编码或是客户编码中很常用，但是很少作为一种压缩技术。

优势：更高性能的列操作

在列式数据存储结构中，对于单列的操作，例如搜索或是聚合，以遍历存放在连续区域的数组的方式实现。这样的操作具有很高的空间局部性，并能有效地在处理器缓存中执行。在行式存储中，相同的操作会更慢因为同一列的值分布在内存中，并且处理器被缓存未命中减低了速度。

假设我们要对图 2 中行式表的销售总和进行聚合，从主内存到处理器缓存的数据传输总是发生在被称为“高速缓存行”（例如 64 个字节）的固定大小的块。行式存储中时常会发生每个缓存行只含有一条“销售”值（4 字节存储），而剩下的字节存放的是该记录的其余字段。对于聚合所需的每个值，每次需要重新访问内存。这意味着行式存储结构中，操作会因导致处理器等待所需数据的缓存未命中而减速。基于列的存储，所有的销售值存储在连续内存中，因此高速缓存行中的 16 个值都是在总和计算所需要的。此外，列存储在连续内存使用内存控制器预读取，从而进一步减少缓存未命中数。

正如上文解释的，列式存储结构也允许高效的数据压缩，这不仅节省了内存空间，同时也因以下原因提升了速度：

- 压缩后的数据可以更快的加载至处理器的缓存中。这是因为限制因素是内存和处理器缓存之间的数据传输，所以性能增益将超过需要的额外解压缩时间。
- 利用字典编码，列字段以位编码的整数顺序存储，这意味着可以对整数判断是否相等（例如在扫描过程中或是联接操作），速度远远快于字符串的比较。
- 在操作符识别压缩的情况下，压缩可以加快扫描和聚合等操作。如果列运行长度编码和许多相同值的和可以通过一个单一的乘法取代，鉴于良好的压缩率，对列中的值求和会快很多。最近的科学研究报告显示，对整数编码压缩后的值进行比较操作比未压缩的值快100到1000倍。

优势：消除额外的索引

很多案例中，列式存储不再需要额外的索引结构，在功能上，按列存储数据相当于给每一列添加了内置的索引。由于内存列存储以及压缩机制，扫描列的速度使得读操作具有非常高的性能，特别对字典压缩而言。在许多情况下，消除额外的索引降低了复杂性，减去了定义和维护元数据的工作。

优势：并行

列式存储利用处理器的多个内核，使得并行操作变得容易。在列式存储中，数据已被垂直分隔，这就意味着不同列的操作可以并行的执行。如果要对多个列进行扫描或者聚合，这些操作可以分配到不同的内核中。另外单列上的操作也可以并行分割成多个部分，可以由不同的处理器内核处理。

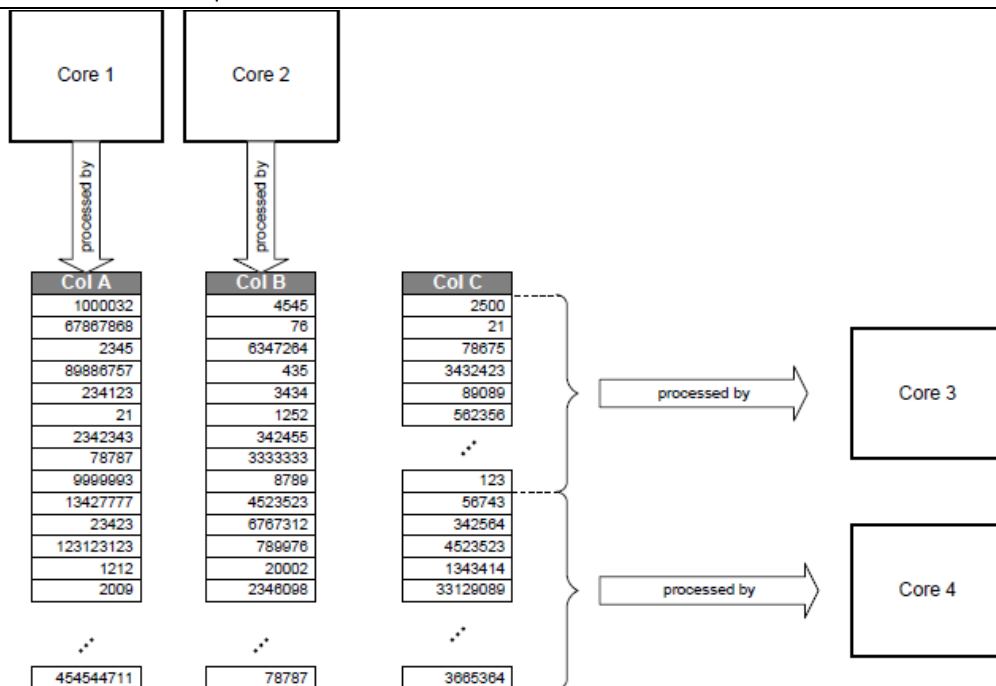


图 3 列式存储并行例子

优势：消除物化聚合

传统的业务应用通过物化聚合来提升读取的性能。这意味着程序开发人员需要定义额外的表，冗余地存放其他表计算出的聚合结果（如求和）。物化聚合的计算和存储或是在每次汇总数据的写操作后，或者是按预定的时间，读取操作则是在每次需要时才进行计算。

随着每毫秒的几个 GB 的扫描速度，内存中的列存储可以高性能地计算大量数据的聚合结果，这在很多情况下消除了物化聚合的需要。

在财务应用中，不同种类的总数及结余通常被物化聚集到不同账簿中，比如总账，应付账款，应收账款，现金总账，材料明细账等。利用内存列存储不再需要这些物化聚合，因为总账和结余可以从会计凭证中很快计算出结果。

消除物化聚集有几个好处：

简化的数据模型。物化聚集需要使用额外的表，这会使得数据模型变得复杂。例如，SAP Business ByDesign 的财务应用，持久化总数和结余以星型模式存储，为总数和结余引入了特定的业务对象，每个对象都伴有一张事实表和几张维度表，如果总数和结余可以一瞬计算出的话，就可以不需要这些表了。一个简化的数据模型，使开发更高效、消除编程错误的来源、增加了可维护性。

简化的业务逻辑。程序需要在每次聚合新增、删除或是修改之后更新其值，或者需要特殊的聚合运行计划，在一定的时间间隔（例如每天一次）更新聚合值。通过消除持久化聚合，就不需要额外的逻辑。

更高级别的并发性。对于物化聚合，每次执行完写操作后，需要获得一个读写锁来更新聚合，这限制了并发性也会导致性能问题。当不需要物化聚合时，只有文档项目会被写入，这可以

聚合值的同时性。利用快速的聚合，值总是最新的，而物化的聚合只在预先定义的时间更新。

2.2 架构概览

图 4（见下图）中的方框给出了 SAP HANA 数据库的概念结构图。在方框的顶端可以看到数据库连接着客户端。一个客户端连接数据库形成了一个会话，通常这里使用的通信是以 SQL 语句的形式。在 SAP HANA 的数据库，每条 SQL 语句在一个事务上下文中处理。新的会话隐式分配到一个新的事务中。事务管理器是协调数据库事务、控制事务分离和跟踪运行中，关闭事务的组件。当一个事务被提交或是回滚时，事务管理器会把该事件通知给参与的存储引擎，让它们来执行必要的步骤，并负责协调持久层完成原子和持久的事务。

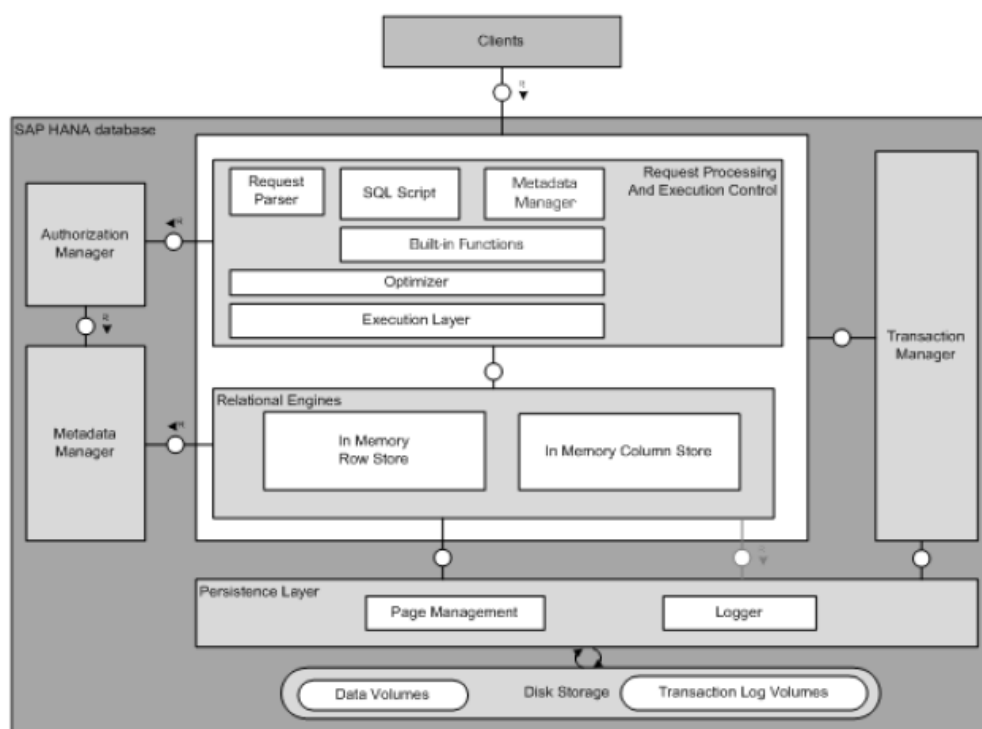


图 4 SAP HANA 高层体系结构

客户端的请求会被一系列的组件分析并执行，称作请求处理和执行控制。图 4 展示了这层的主要功能：请求解析器分析客户请求，然后分发到相应负责的组件。例如，事务控制语句会被发送到事务管理器；数据定义语句会传到元数据管理器而对象调用则到对象存储器。数据操作语句转发至优化器，生成一个优化的执行计划给执行层。

SAP HANA 的数据库有内置的支持特定领域模型（如财务规划），提供脚本功能，允许在数据库内运行应用程序特定的计算。SAP HANA 有自己脚本语言，SQLScript，设计用来优化和并行操作。SQLScript 是基于无副作用的函数，用 SQL 查询语句对表进行集操作。

SAP HANA 数据库也包含一个名为规划引擎的组件，允许金融规划程序执行在数据库层的基本规划操作。一个基本的操作是创建一个新版本的数据集，应用过滤和转换，作为原有数据的拷贝。例如，以上年的数据副本创建新的一年的规划数据，这就要求根据年份过滤并且更

新时间维度。另一个例子是，根据分解函数，规划操作把目标值从高聚合层次降低聚合。

SAP HANA 中的功能诸如 SQLScript 和规划操作，都是以内置函数中公共的基础结构实现的。

元数据可以通过元数据管理器访问。SAP HANA 元数据由一系列的对象组成，如关系表、列、视图、索引的定义，SQLScript 函数定义和对象存储元数据。所有这些类型的元数据存储在一个共同所有 SAP HANA 的数据库存储目录（内存行存储，内存列存储，对象存储，基于磁盘）。元数据以行形式存在表中。SAP HANA 的功能，如下面的章节中所描述的数据库多版本并发控制，事务支持，也可用于元数据管理。在分布式数据库系统的核心中，元数据在服务器之间共享，元数据是如何存储和共享对使用元数据管理器的组件是透明的。

由于行式表和列式表可以在一个 SQL 语句结合使用，相应的引擎必须能够消费相互创建的中间结果。两种引擎的区别存在于它们处理数据的方式：行式存储操作符用迭代器一次处理一行数据，而列式存储操作（如扫描，聚合等等）需要整列都放在连续的内存空间中。为了交换中间结果，行式存储可以在内存中，把结果物化成一行完整记录提供给列式存储；而列式存储也能通过利用行式存储所需的迭代器，公开显示结果。

持久层负责事务的持久性和原子性，能保证数据库重启后恢复到最近的提交时间点以及事务完全执行或者完全取消。为了实现这一目标，持久层使用预写日志，影子分页和保存点相结合的有效途径。持久层提供了写数据和读数据的接口，同时还包含了 SAP HANA 管理事务日志的记录。当数据通过持久层接口写入时，持久层隐式的增加一条日志记录，或当显示调用日志接口时，也会新增记录。

SAP HANA 数据库组件通过激活授权管理器来检查用户是否有相应的权限执行请求的操作。SAP HANA 允许授予用户或角色权限。权限授予对特定对象（比如表，视图，SQLScript 函数等等）执行特定操作（例如增删改查等）的权利。SAP HANA 数据库支持分析权限，用来展示分析过滤器或向下钻取分析语句。分析权限授予访问某些维度属性值的组合，这可以用来限制访问数据立方中维度属性区域为“美国”和年份为“2010”的值。由于分析权限是在定义在维度属性值上，而不是元数据中，因此它们在查询执行过程中动态评估。

用户进行身份验证可以由 SAP HANA 的数据库本身完成（用户名和密码登录）或委托给外部认证机构，如 LDAP 目录。

2.3 SAP HANA 数据库概念：表，模型和视图处理

SAP HANA 的数据库是一个非常强大的数据库系统，但它需要一定的了解，并正确使用以获得良好的性能。在这节中，我们会探索一些 SAP HANA 数据库的关键概念，并且分析如何使用建模得到良好的结果。

2.3.1 表，视图和星型架构

SAP HANA 允许把你的数据以表和视图的方式建模。表是表格数据结构，每一行确定一条具体的实体，每列有一个唯一的名称。一行中的列称为实体的属性，“属性”在本文档中有着不同的含义。它可能是指一个表列、特定的数据字段的表行、一个数据字段的内容，通过上下文文明确各自的含义。

视图是以数据表为蓝本，为特定目的的组合和选择。视图经常是可读的表，如读取表的操作也可以读取视图。例如，您可以创建只从表中选择一些列的视图，或根据过滤模式选择一些列和行的视图。在本文档中，会经常提到术语架构或数据库架构，架构是指由 SQL 描述的数据库结构，其中的表和表之间的关系。

在分析型应用中，星型架构是常见的模式：事实表是业务数据的清单，通常和主数据表相连。这些连接的表称为维度表。维度表围绕一个事实表的结构常被称为星型架构，因为用图形展示时看起来像星星。在 SAP HANA 中，可以通过使用属性视图围绕分析视图或是计算视图，来构成星型结构。

2.3.2 SAP HANA 模型视图

在我们继续讨论如何在 SAP HANA 中建模之前，很有必要了解每一种不同 SAP HANA 模型视图类型和功能。

属性视图

属性视图用于定义表之间的连接，并以合理的方式告知系统将要进行的连接。它们也可以用来从表中选择行或列的子集。

星型结构中，属性视图的一个应用是联接多个表从而构成一个单独维度表。由此生成的维度表可以通过分析视图联接事实表，使数据变得有用。例如，属性视图可以将员工联接到组织单位，然后通过分析视图与销售业务数据相连。

分析视图

典型的分析视图定义在至少一张含有交易数据的事实表，使用分析视图，你可以创建一系列的度量值（有时称为主键字段），添加属性和联接属性视图。

分析视图利用 SAP HANA 的运算能力计算汇总数据，例如每个国家销售的汽车数，或是每天的最大功耗。它们定义在至少一张事实表中，如一辆车对应一条数据或是一个电力表一条数据，或者，更一般来说，含有一些业务交易记录的形式。事实表可以和一个分析视图联接来访问更详细的数据。分析视图可以定义在一张表或是联接表上。

分析视图可以包含两种类型的属性（或列），称为度量字段和主键。度量字段是在聚合中必须定义的属性。如果分析视图是在 SQL 语句中使用，那么度量字段必须聚合，例如使用 SQL 的函数 SUM(列名)、MIN(列名)、MAX(列名)。正常的属性可以作为普通列处理，而没有必要进行聚合。

计算视图

计算视图合成了其他的视图，从本质上讲，他们是基于联接或合并的两个或两个以上的数据流或调用内置的通用 SQL 函数。

计算视图被定义成图形化视图或者脚本视图，但不是 SQLScript（有例外，见下文），主要根据它们创建的方式。它们可以同分析视图一样的方式使用，但是和分析视图不同的是，计算视图中可以联接多张事实表。计算视图总是含有不止一个度量值。

图形化视图可以通过 SAP HANA Modeler 的图形建模特性而建立。脚本视图则是由 SQL 语句

序列创建。

如前所述，计算视图一般不使用 SQLScript 创建，但也有例外，含有如下属性的 SQLScript 可以用来建造计算视图

没有输入参数

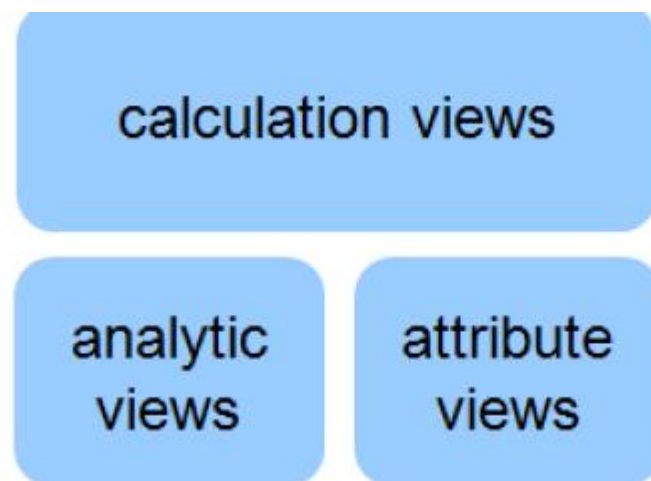
总是只读（不改变数据库）

没有副作用

2.3.3 SAP HANA 视图处理

对于基本了解 SAP HANA 如何处理视图是必须的，这样你可以保证在数据库中，数据传输最小化。

下面的图表说明了一个系统的简化视图。



从图中，我们看到 SAP HANA 有三种类型的视图，它们都在建模的要求下使用。

- 计算视图：被用于分析视图和属性视图的顶端，来进行分析视图和属性视图无法执行的复杂操作。
- 分析视图：用于“基于星型模式”的计算和聚集或类似操作。
- 属性视图：用于所有类型的联接。

SQL 优化器根据涉及的模型和查询语句决定调用数据库系统函数的最佳方式。

此图是简化了的图，例如，一个分析视图含有计算过的属性或包括了一个属性视图含有计算后的属性，也会变成一个计算视图。这在建模中也应该被考虑，因为它可能对数据模型的性能有很大的影响。

建模指南-最小化数据传输

正如文档中已多次强调，建模中的最基本目标是最小化数据传输。这适用于内部的三个，SAP HANA 数据库视图和视图之间，也在 SAP HANA 和终端用户之间。例如，终端用户永远不想看到一百万条的数据，他们永远不可能理解这么多的信息或者以有价值的方式使用。这意味着，数据应当在离开数据层钱，尽可能的聚合以及过滤到一个可控的大小。

当决定报告的记录时，一个最好的办法就是思考在“集和层次”而非“记录层次”。一个数据集可以汇总一个地区，一个日期，或者其他一些组的数据，以尽量减少视图之间数据传输量。

基本建模规则：一个例子

在这个例子中，需要两张事实表由它们的主键而联接起来，这推动了数据查询是如何执行的。正如前文所提到的，以数据集的方式思考而非数据行（或记录）。左边的图展示了一个糟糕的设计，因为它通过主键进行联接，这样做，在查询运行时就不得不访问每条记录。这在运行时代价是很大的，由于在计算视图中联接数据。

在两张分析视图会产生很多记录的情况下，这种联接必须避免并且用合并代替。联接可以被使用当其中一张分析视图生成少数记录（少于一百万条）至计算视图。

在把数据从分析视图传输到计算视图之前，数据必须最小化- 这可以用投影实现。利用投影，分析视图生成的记录数可以通过分组和过滤字段而减少。在创建计算视图可以采纳该意见来避免性能问题。

右边的图像展示了一个更有效的创建视图的方式。请注意两个独立的分析视图，最初按地区分组，然后通过合并放在一起，这比联接在行层次上更有效。

海量数据使用集合处理

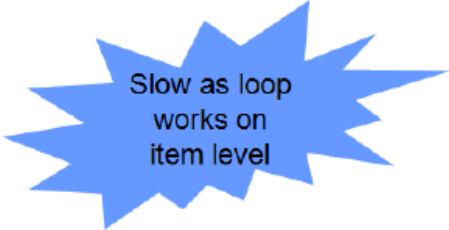
上节中已提到，使用“集合方式”而不是“行方式”来处理数据，总是能获得较好的性能。在本节中，我们将进一步研究。

下文中的两条查询都最终访问同一数据。第一条查询在第一行设置了限定，然后必须再次查询所有数据集的其余部分。

下面的代码是一个糟糕的查询例子：

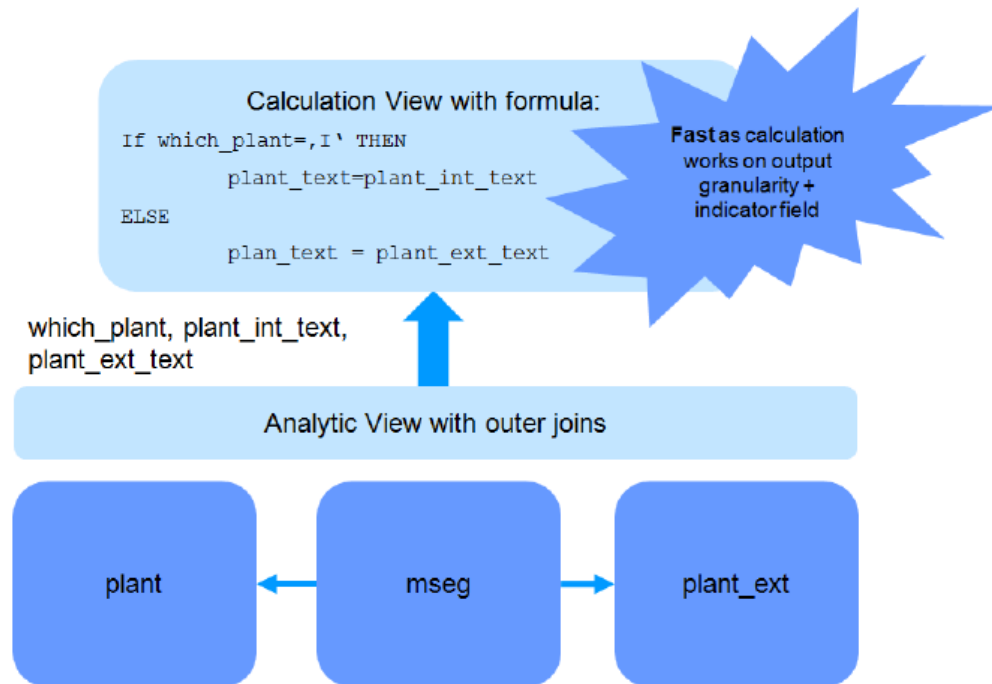
```
matmoves = SELECT * FROM MSEG
FOR EACH matmove in matmoves
  IF matmove.whichPlant = „I“ THEN
    plant_text = SELECT plant_text FROM WERKS WHERE id=matmove.plant
  ELSE
    plant_text = SELECT plant_text FROM WERKS_EXT WHERE id=matmove.plant
  END
NEXT
```

[pseudocode]



Slow as loop
works on
item level

下面的查询是一个以更有效的方式获得相同数据的例子。



在建模时避免使用循环是很重要的，因为它会导致性能问题，尤其当要循环的表中数据量很大时（数以亿计的数据项）。

3 教程

3.1. 使用 SAP HANA studio

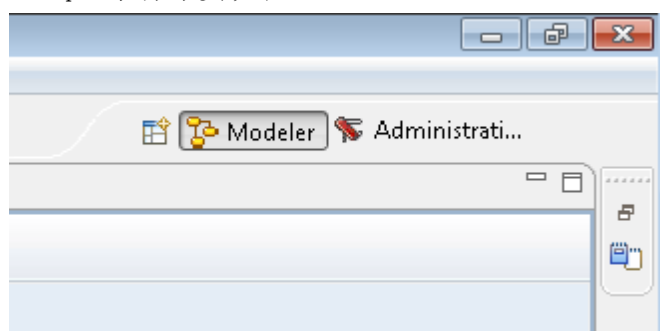
SAP HANA studio 是为那些希望创建数据模型和存储过程，或操作交互式表格和问题查询的开发人员使用的工具。在下面的教程，我们假设如下：

你可以通过 TCP/IP，从工作站访问运行在主机名"hana.mynetwork.com"的 SAP HANA。

实例编号为 99

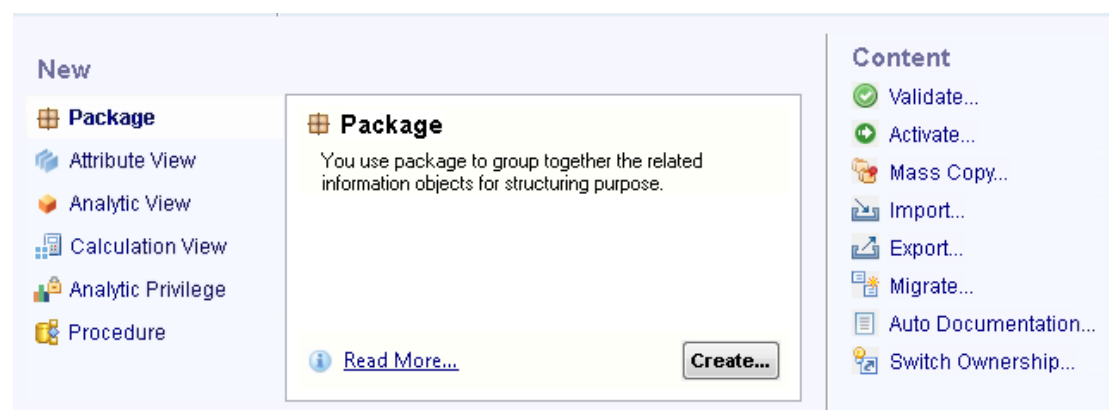
你的工作站已经安装了 SAP HANA studio。为了使用 SAP HANA studio，你需要知道 SAP HANA 账户用户名和密码，这些信息可以从你的数据库管理员那里得到。请注意，某些模式(schema)的访问会受到限制。

请打开您工作站上的 SAP HANA studio，它提供了两块主要的工作区域，被称为透视图 (perspective)，可以在 studio 窗口的右上角进行选择。你经常会用到的透视图是建模透视图，而管理控制台透视图比较少使用。对 Eclipse 熟悉的读者会发现这两种透视图都已经在 Eclipse 框架中实现了。



现在单击右上角相应的区域来切换到建模透视图，或者从 **Window ► Open Perspective ► Modeler** 下拉框中选择。这个透视图提供了创建、编辑数据模型和存储过程的功能，可以在 Business Objects Explorer 的分析程序和经过验证的、支持 MDX, SQL 或 BICS 的产品中使用。

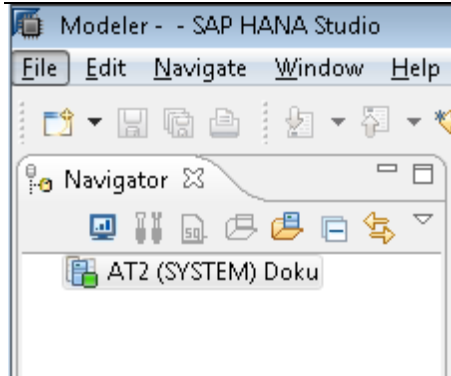
请注意，当你第一次打开建模透视图时，迎接你的是快速启动菜单，你可以利用这个菜单快速访问建模试图的功能。如果你关闭了快速启动菜单，可以通过选择 **Help ► Quick Launch** 重新打开。



在你操作 SAP HANA 数据字典、创建或删除模式、导入导出数据、或是用 SQL 查询数据库时，管理控制台透视图是非常有用的。

利用 SAP HANA studio 注册 SAP HANA 设备

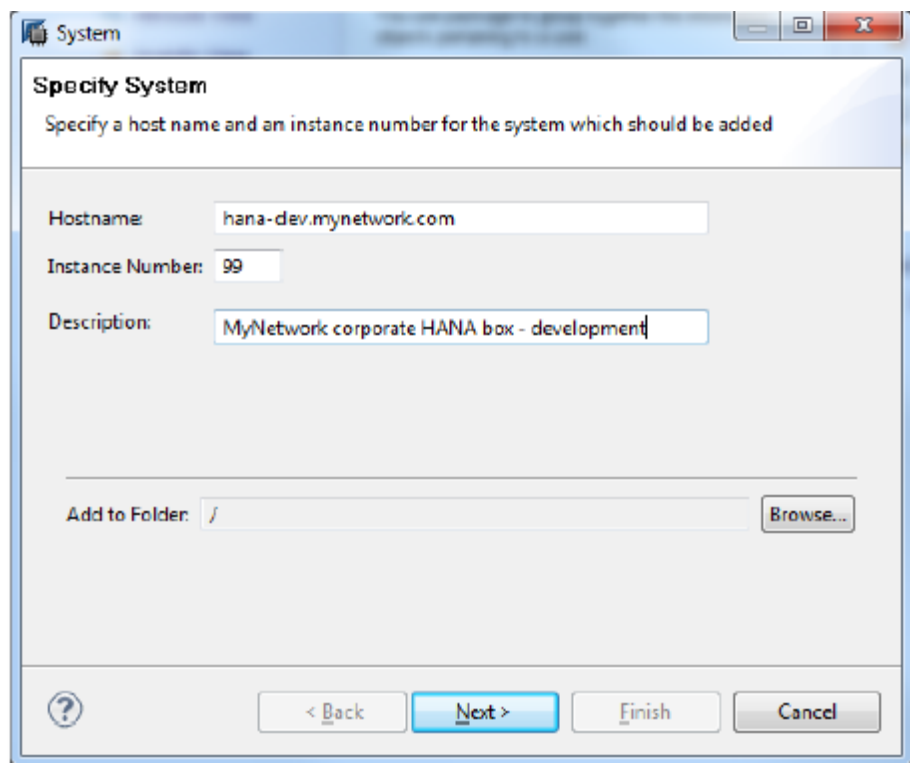
在 SAP HANA Studio 中主窗口中有几个子窗口，称为视图，最左边的一个是导航栏(Navigator)。



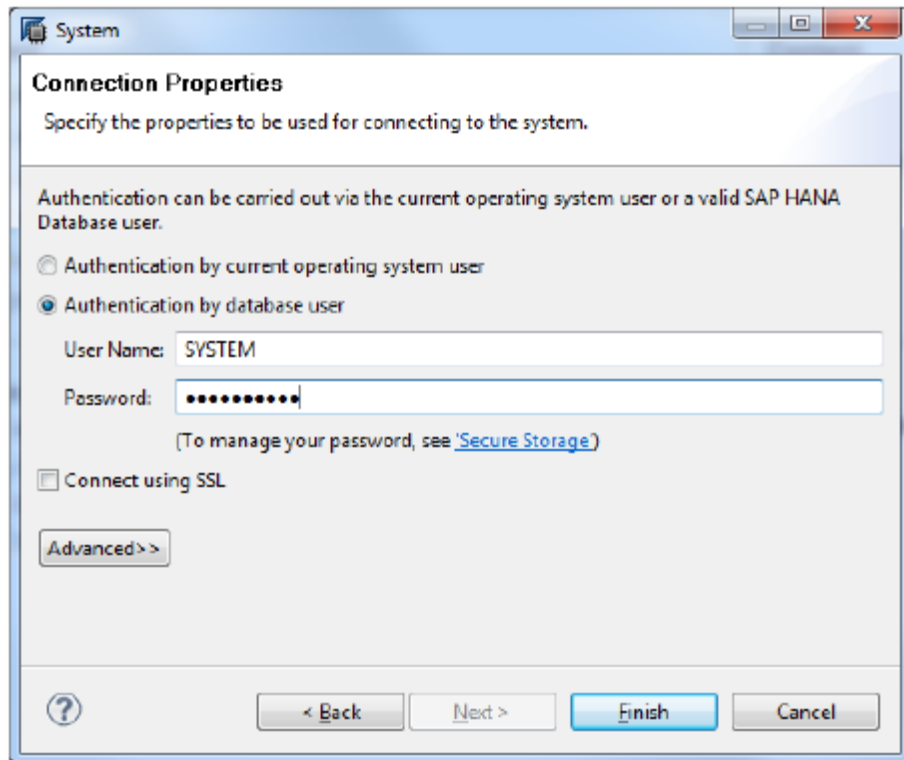
在导航栏中右键单击弹出上下文菜单，从该菜单中选择“*Add System*”，在出现的对话框中输入以下信息：

- Hostname: hana-dev.mynetwork.com,
- Instance Number: 99
- Description: MyNetwork corporate SAP HANA box – development

单击“NEXT”继续。



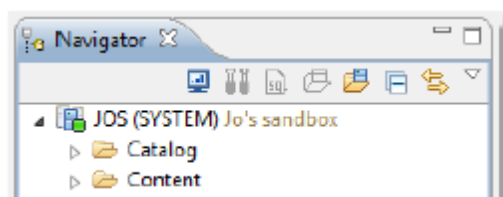
请输入系统管理员给你的系统名或者用户名、密码，然后单击“*Finish*”。如果输入的数据是正确的，一个半展开的树形结构将出现在导航栏中，以系统名作为顶级节点的名称。双击“*Catalog*”，在该节点下你会发现节点“*Authorization*”和“*Public Synonyms*”，以及几个代表数据库模式的节点。



注意，SYSTEM 是 HANA 系统的主要管理用户，通常，你应该使用自己的用户名。

在导航栏视图中，你可以看到一个新的节点名为“*HANA (SYSTEM) MyNetwork corporate SAP HANA box – development*”，单击旁边的白色三角可以看到系统中的内容。其中有两个子节点，“*catalog*”和“*content*”。第一个节点代表了 SAP HANA 数据字典，例如所有的数据结构，表和可以使用的数据。“*Contents*”节点表示设计阶段的元库，含有所有 modeler 创建的模型。物理上，这些模型都存储在数据库的表中，在“*catalog*”下可以看到。“*content*”节点只是提供了一个对同一个物理数据的不同视图。

在下面的截图中，我们将使用一个不同的系统，它的导航栏节点标记为“*JOS (SYSTEM) Jo’s sandbox.*”



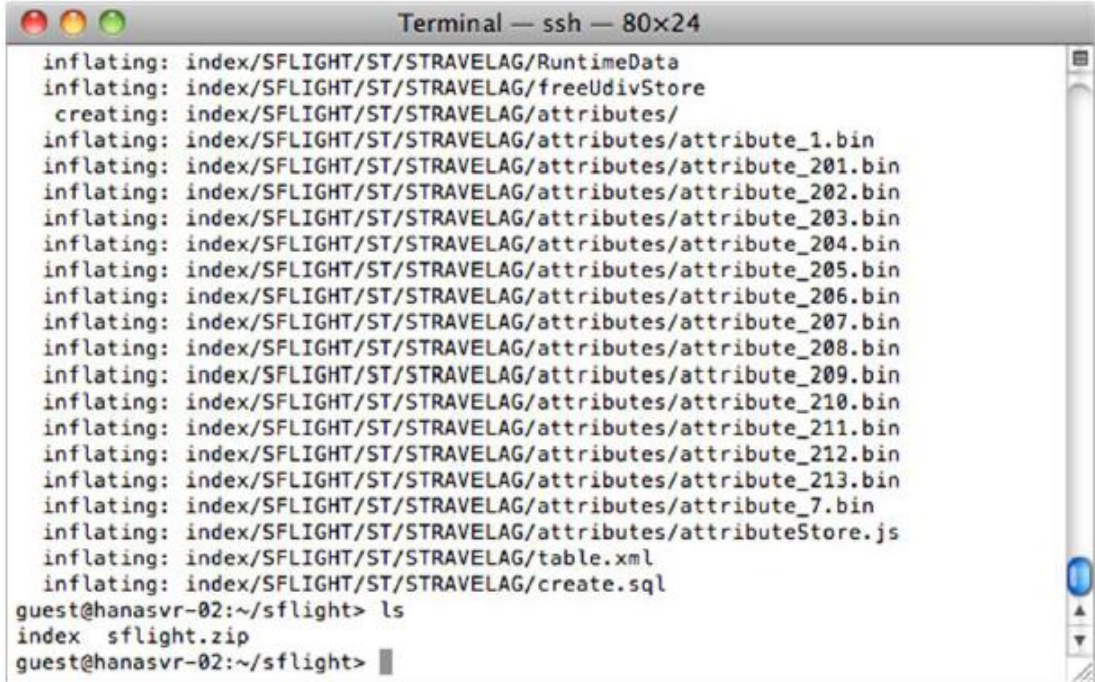
加载示例数据到 SAP HANA

在下面的章节中，我们将使用的数据模型和数据可以从 <https://sapmats-de.sap-ag.de/download/download.cgi?id=UJB3QSF98GSHX6ZADOB90HUWAMDPOX9IC00Q89UPLGPO926JBH>

下载得到。

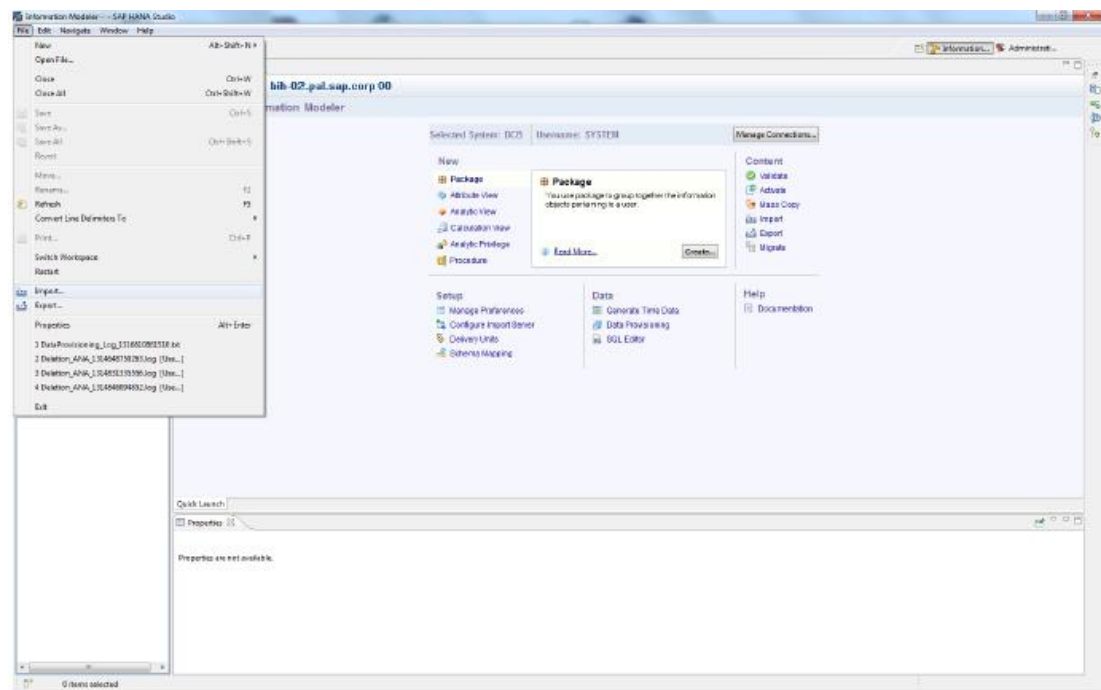
请先下载文件至 SAP HANA 设备中的目录，例如/home/guest/sflight，然后用命令 **unzip sflight.zip** 进行解压缩。

你现在应该可以看到新的目录名为“*index*”。

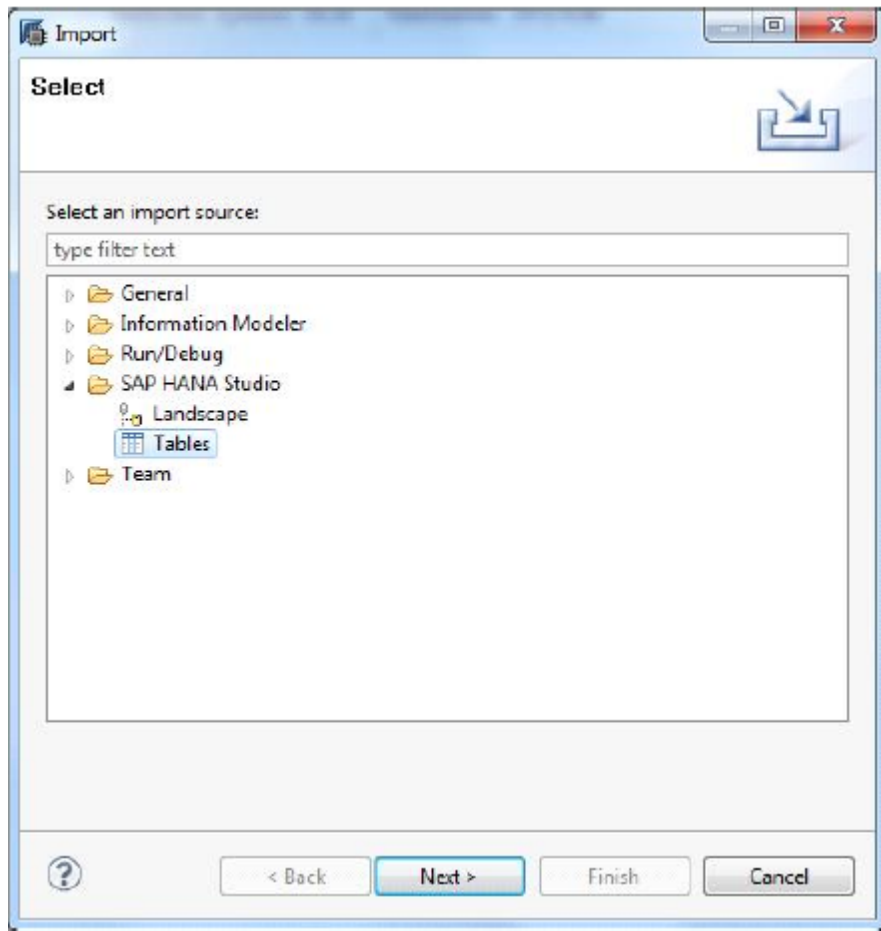


```
Terminal — ssh — 80x24
inflating: index/SFLIGHT/ST/STRAVELAG/RuntimeData
inflating: index/SFLIGHT/ST/STRAVELAG/freeUdivStore
creating: index/SFLIGHT/ST/STRAVELAG/attributes/
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_1.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_201.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_202.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_203.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_204.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_205.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_206.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_207.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_208.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_209.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_210.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_211.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_212.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_213.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attribute_7.bin
inflating: index/SFLIGHT/ST/STRAVELAG/attributes/attributeStore.js
inflating: index/SFLIGHT/ST/STRAVELAG/table.xml
inflating: index/SFLIGHT/ST/STRAVELAG/create.sql
guest@hanasvr-02:~/sflight> ls
index sflight.zip
guest@hanasvr-02:~/sflight>
```

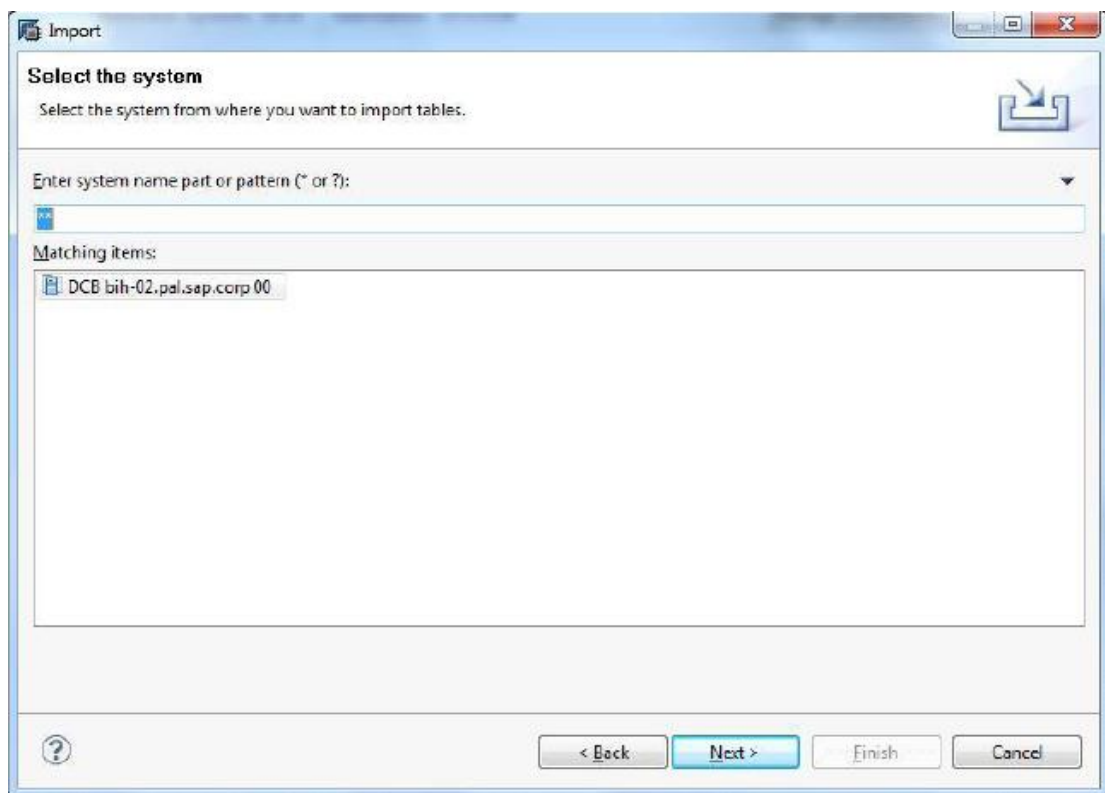
打开 SAP HANA studio，然后从主菜单中或者快速启动菜单选择 **File ► Import**。



在弹出框中，定位到 **SAP HANA Studio ► Tables**，单击“NEXT”。



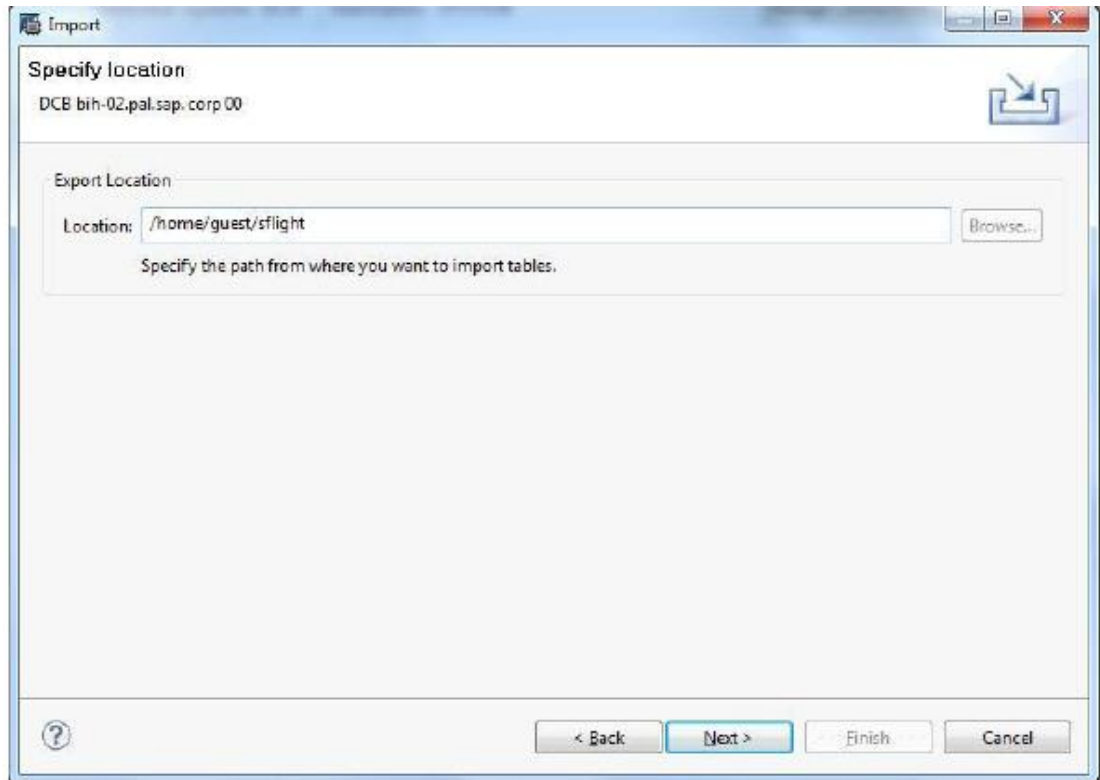
可能会提示您选择需要导入的 SAP HANA 数据库系统。选择的相关系统，并单击“NEXT”。



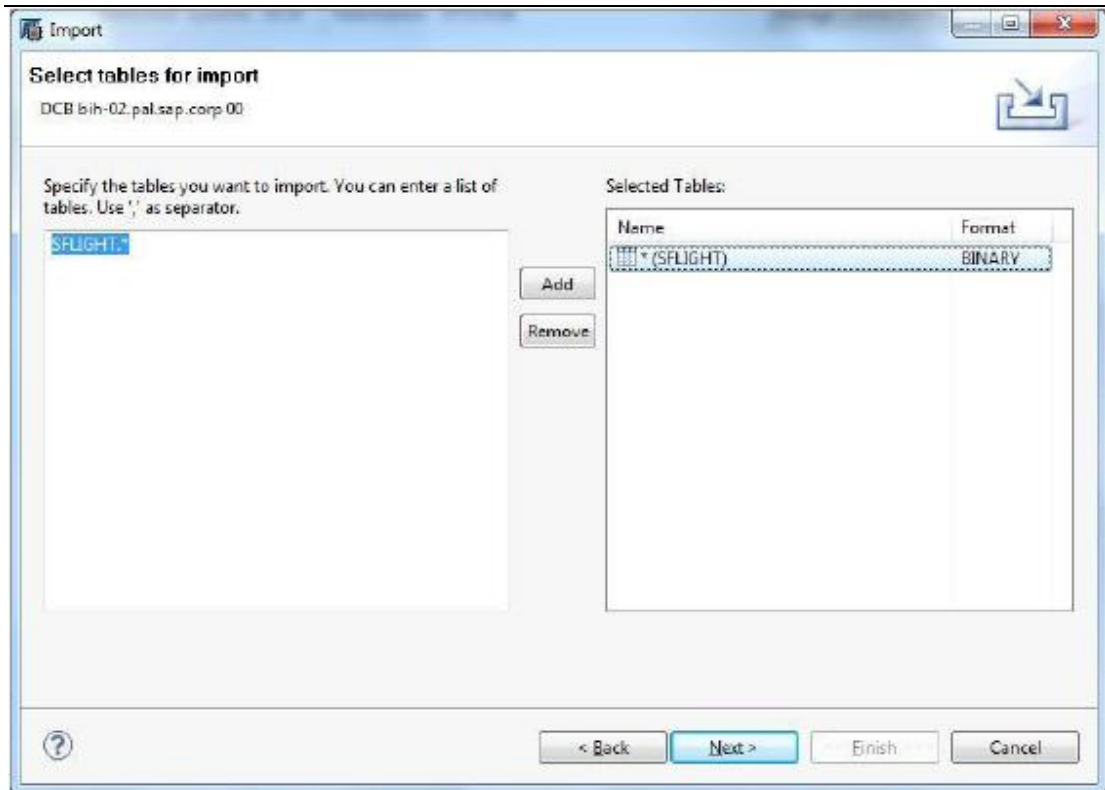
输入上传目录的路径，例如/home/guest/sflight，单击“NEXT”。

每个 SAP HANA 系统都有一个 *System ID* (SID)，从这个我们可以知道系统管理员的用户名，形式为<SID>adm，例如如果 *System ID* 是 AT2，<SID>adm 用户名就是 AT2adm。

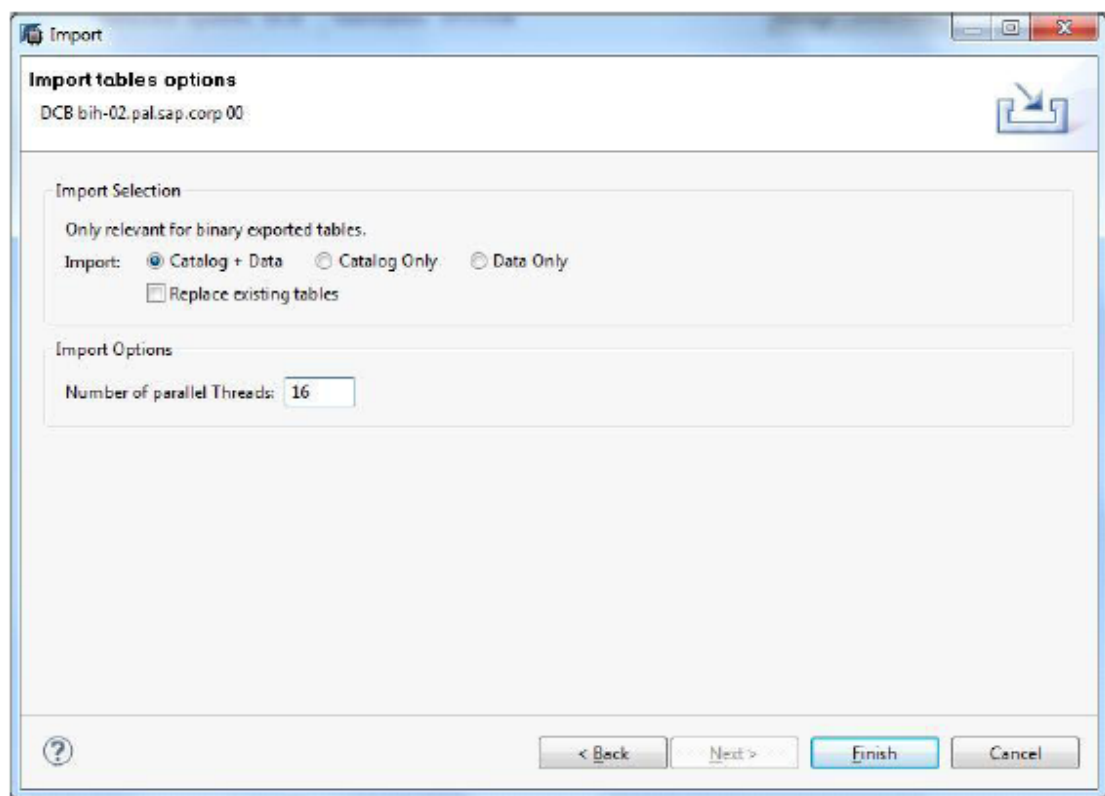
请注意，SAP HANA 系统<SID>adm 用户需要至少有读导入数据的目录的权限。



在接下来的截图中，输入“SFLIGHT.*”（代表 SFLIGHT 模式下所有导入目录的表）到左边的文本区域，然后单击 Add 按钮。一条新的记录*(SFLIGHT)出现在右边的表中，单击“NEXT”。



最后，勾选上单选框“Catalog + Data”并输入导入所需的并行执行线程数。最佳数量等于或小于您的 SAP HANA 的服务器核心数，但也取决于你有多少用户共享系统。您选择的线程数越多，你的上传速度也越快（同时也影响了更多的其他用户）。



经过短短几秒钟（对于 16 个线程，应该不到一秒钟），您可以刷新目录（点击目录中的导航

区，然后按 F5) 可以看到新导入的 SFLIGHT 模式。

我们稍后将学习如何导入表和数据，以及如何显示它们。

3.2 如何使用 SAP HANA Modeler

在接下去的部分，我们会学习如何使用 SAP HANA Modeler 根据我们导入的数据集来创建不同的视图。我们从一个简单的分析视图例子开始，然后进一步结合属性视图和分析视图来形成星型架构。我们将学习如何集中于单一的客户端的数据，从属性中过滤数据以及添加从其他属性计算的属性。

要了解 SAP HANA Modeler 的详细功能，请参阅 [SAP HANA Modeling Guide](#)。

四步构建高性能数据模型。例子的顺序被用来反映应该遵循的，正确的通向更复杂场景的发展路径。只要有可能，你应该按照这里介绍的顺序操作，以产生最佳性能的模式。

总的来说，你首先应该从分析视图开始。如果分析视图没有提供足够的功能，就增加属性视图来构成星型架构；如果仍没有满足要求，或者尝试使用计算视图，更灵活但也更复杂。这之后，在最终使用脚本视图前试下图形化视图。欲了解更多详情，请参阅章节 4.3.5。

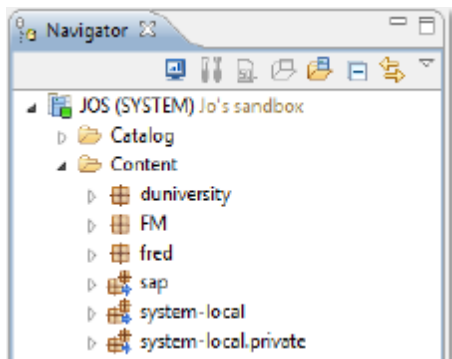
我们的场景是基于航班订票。我们将要处理的问题包括：

- 每个旅行社能产生多少利润？
- 我们如何把旅行社的利润和旅行社的详细信息结合起来？
- 如果以某一特定货币订票，所给的折扣能生成多少利润？

要继续我们的第一个例子，请打开您工作站上的 SAP HANA Studio 并转到 Modeler 工作区。

3.2.1 如何新建分析视图

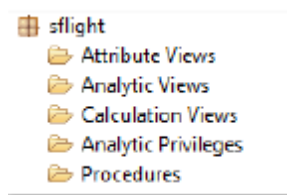
接下去的步骤将会教您如何使用 Modeler 来新建分析视图模型。让我们先在元库中定义一个存放模型的空间。首先单击“content”节点展开，在该节点中，你应该能看到一系列的包节点。



右击“content”节点，然后选择 **New ► Package**，在弹出的窗口中，请输入用户名和新的模型包的描述，单击 **ok** 保存。

在 **content** 下有个新的包节点叫 **sflight**，单击白色小三角可以显示它的内部结构。有些子节

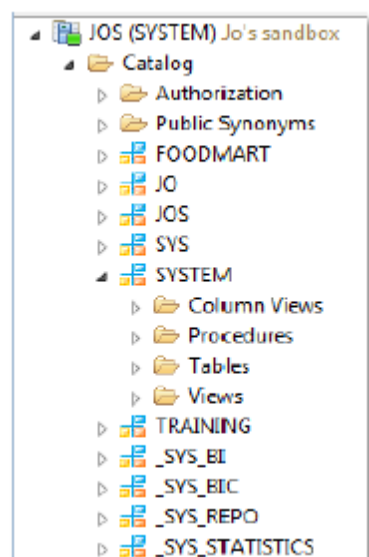
点是很明显的：属性视图文件夹是用来存储属性视图、分析视图文件夹则储存分析视图、计算视图文件夹就放计算视图。这三种视图类型在前面都已讨论过。分析权限（*Analytic Privileges*）用来绑定几个包和模型，并可以用同一组证书访问它们。节点过程（*Node Procedures*）用来存放 SQLScript 存储过程。



现在我们准备好了建第一个视图，但是在我们继续之前，需要了解几张我们将要使用的表。

3.2.2 查看表

我们之前加载的表中数据是关于航班，旅行社，顾客，机票，预订和机上饮食。表和表中的内容都是从教学例子 SFLIGHT 中获得，在每个 SAP IDES 系统中都有，该系统是预配置和预加载的 SAP ERP 系统，专门用于教育目的。请注意，下载包中只包含了 SFLIGHT 表一部分子集。

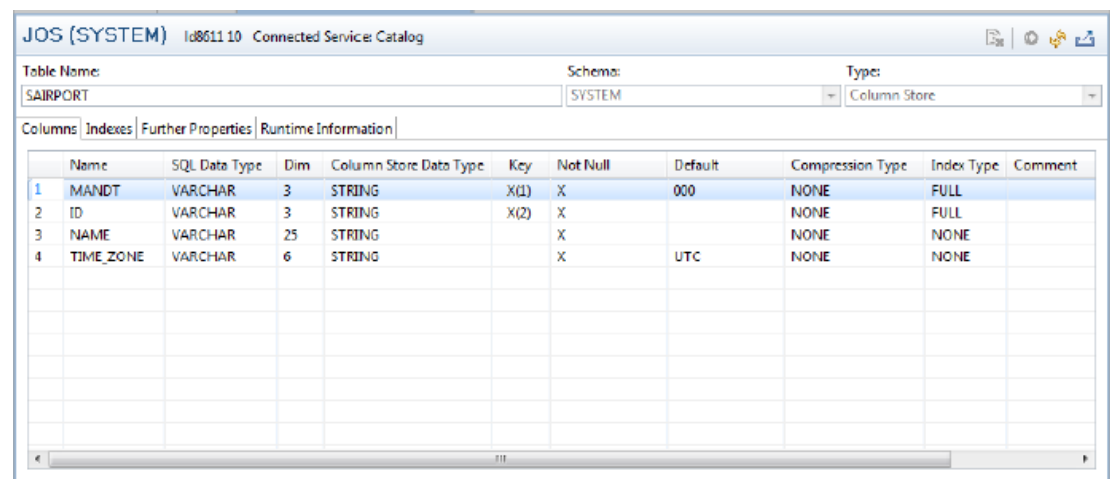


需要查看表，请单击导航栏视图中 **Catalog** 旁的小三角，可以看到不同的模式。请单击 **SYSTEM** 旁的三角，这是数据表导入的模式节点。还有其他一些重要的模式：**_SYS_REPO** 存储出现在 **content** 下的设计时模型；**catalog** 下的 **_SYS_BIC** 保存这些模型运行时生成的对象。请单击 **tables** 旁的三角来查看 **SYSTEM** 模式中所有表的清单。以下的 **SFLIGHT** 表可供使用：

- **SAIRPORT** –机场名称和时区信息的主数据表
- **SAPLANE** –飞机类型，电子信息的主数据表。例如座位数，重量，油箱容量和运行速度
- **SBOOK** –包含所有预订机票和他们的详细资料如航班日期，航班号，价格，客户ID，并售出的门票，旅行社的ID的事实表。
- **SBUSPART** –业务伙伴信息的主数据表
- **SCARR** –航空公司信息（名称，货币，网址）的主数据表
- **SCURR** –货币转换因子表
- **SCURX** – 用于显示货币值的小数点位数

- SCUSTOM –乘客信息主数据表
- SDESSERT –飞行过程中的甜品信息主数据表
- SFLIGHT -每个航班信息，例如电子邮件，日期，价格，机型，座位主数据表
- SMACOURSE – 飞行中提供的主食主数据表
- SMEAL –飞行中可用的膳食信息的主数据表
- SMEALT –不同语言的膳食信息主数据表
- SMENU –可用的膳食组合信息主数据表
- SSTARTER –飞行中提供的主食主数据表
- STICKET –机场发出的机票的主数据表
- STRAVELAG – 旅行社信息的主数据表

右击表节点SAIRPORT，然后选择Open Definition来展示表的结构。在主界面中，你会看到一个表格，显示了表的技术细节。从显示中我们能了解表名(SAIRPORT)，模式（SYSTEM）以及存储类型（列式存储）。除此之外，还描述了不同的列名或属性。欲了解该视图的全部功能，请参阅SAP HANA Modeling Guide.

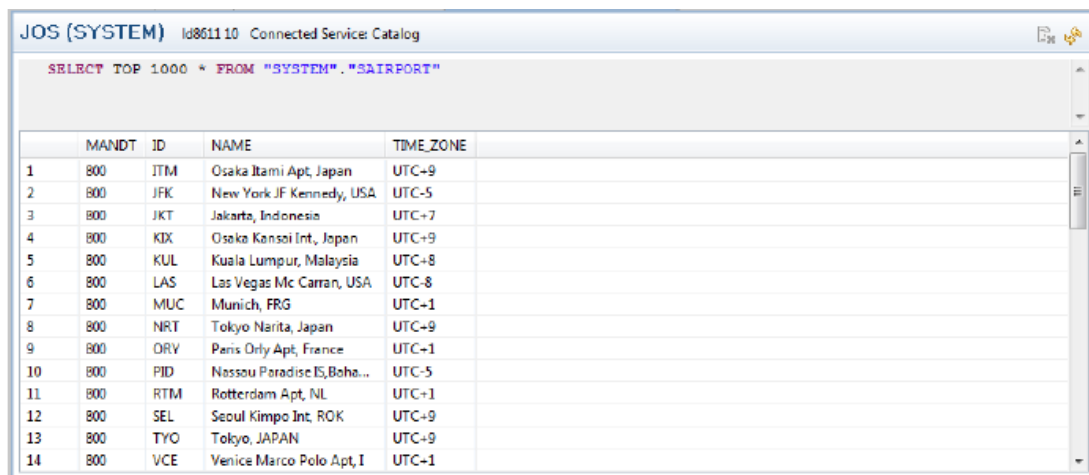


Name	SQL Data Type	Dim	Column Store Data Type	Key	Not Null	Default	Compression Type	Index Type	Comment
1 MANDT	VARCHAR	3	STRING	X(1)	X	000	NONE	FULL	
2 ID	VARCHAR	3	STRING	X(2)	X		NONE	FULL	
3 NAME	VARCHAR	25	STRING		X		NONE	NONE	
4 TIME_ZONE	VARCHAR	6	STRING		X	UTC	NONE	NONE	

很多表都含有MANDT列，包含了客户端或租户的编号。这列在SAP ERP系统中用来区分属于不同公司的数据，例如控股公司和附属公司的数据。除此之外，我们的机场表有一列三个字母的IATA机场代码、机场名字以及他们的时区。表的主索引包括主键MANDT和ID。在这些例子中使用的数据集包含来自客户端200和800的数据，请检查您的SAP HANA安装包和SAP ERP系统的客户端。

要预览表中的数据，请右击表节点SAIRPORT，选择Open Content。在主界面中你会看到新的标签显示了表的几行以及用来访问数据的SQL语句。你可以复制拷贝SQL语句到其他程序中或者我们后面会讨论的SQL编辑器。

请用点时间查看SFLIGHT表，我们很快将会用到其中的几张。



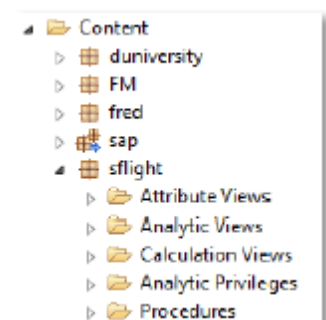
The screenshot shows the SAP HANA Studio interface. At the top, it says 'JOS (SYSTEM) Id861110 Connected Service: Catalog'. Below this, a SQL query is entered: `SELECT TOP 1000 * FROM "SYSTEM"."SAIRPORT"`. The result is displayed in a table with the following columns: MANDT, ID, NAME, and TIME_ZONE. The table contains 14 rows of data, listing various airports and their time zones.

	MANDT	ID	NAME	TIME_ZONE
1	800	ITM	Osaka Itami Apt, Japan	UTC+9
2	800	JFK	New York JF Kennedy, USA	UTC-5
3	800	JKT	Jakarta, Indonesia	UTC+7
4	800	KIX	Osaka Kansai Int, Japan	UTC+9
5	800	KUL	Kuala Lumpur, Malaysia	UTC+8
6	800	LAS	Las Vegas Mc Carran, USA	UTC-8
7	800	MUC	Munich, FRG	UTC+1
8	800	NRT	Tokyo Narita, Japan	UTC+9
9	800	ORY	Paris Orly Apt, France	UTC+1
10	800	PID	Nassau Paradise IS, Baha...	UTC-5
11	800	RTM	Rotterdam Apt, NL	UTC+1
12	800	SEL	Seoul Kimpo Int, ROK	UTC+9
13	800	TYO	Tokyo, JAPAN	UTC+9
14	800	VCE	Venice Marco Polo Apt, I	UTC+1

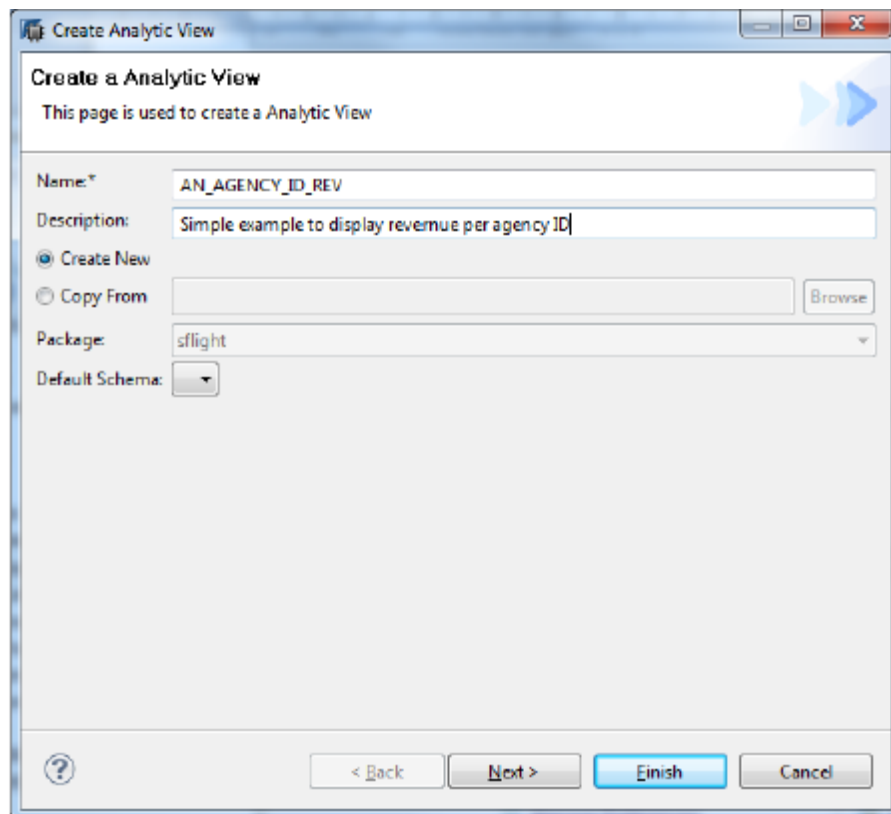
3.2.3 创建分析视图

作为我们的第一个例子，我们会在SBOOK表上创建一个分析视图。假设我们想要报告每个旅行社总的收入。为了简单起见，我们将显示旅行社的数字标识，而不是全名。我们稍后将处理如何得到文本名字。

在导航栏视图中，请展开你数据库中的content节点，然后展开sflight节点，右击Analytic Views，选择New ► Analytic View。

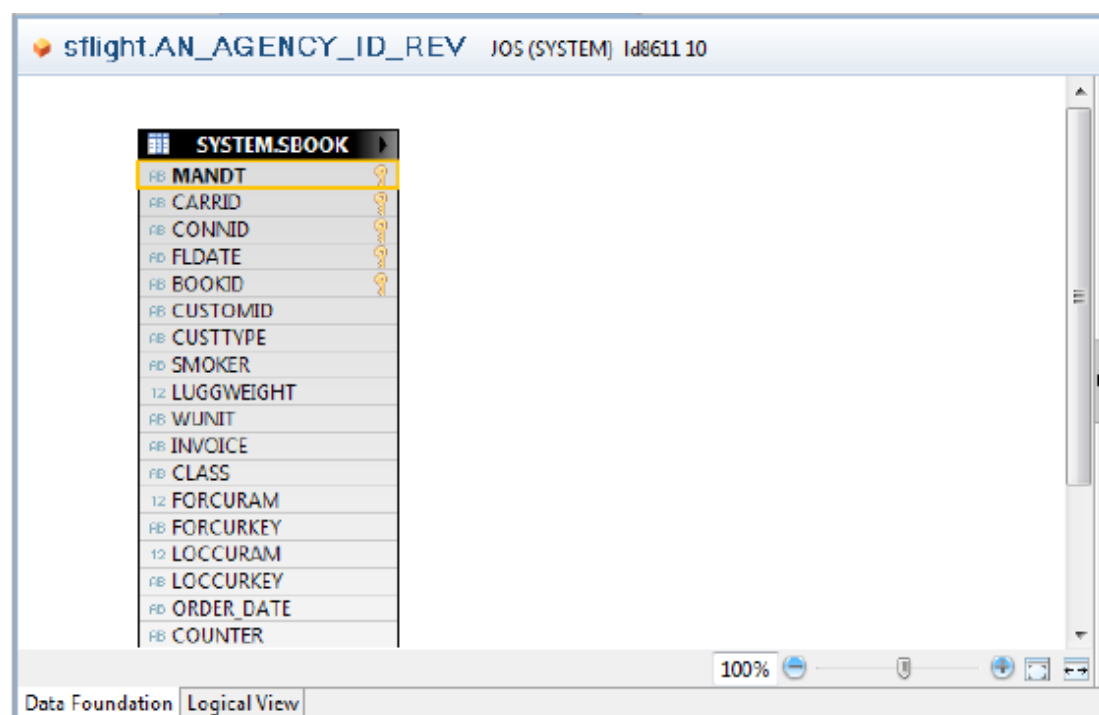


在弹出窗口中，输入你的分析视图的名字和描述，这里我们用名字AN_AGENCY_ID_REV，单击FINISH保存你的分析视图。



在Modeler的主界面中，出现了一个名为sflight.AN_AGENCY_ID_REV的画布。在画布上是空的，因为我们没有定义任何在我们的分析视图中使用的表。现在让我们添加一个表。

在导航栏界面找到SBOOK节点，位于Catalog/SYSTEM/Tables下。拖拽SBOOK节点到空的Modeler画布上。在Modeler的画布上，现在显示包含的表名称和所有其属性和列的一个矩形框。



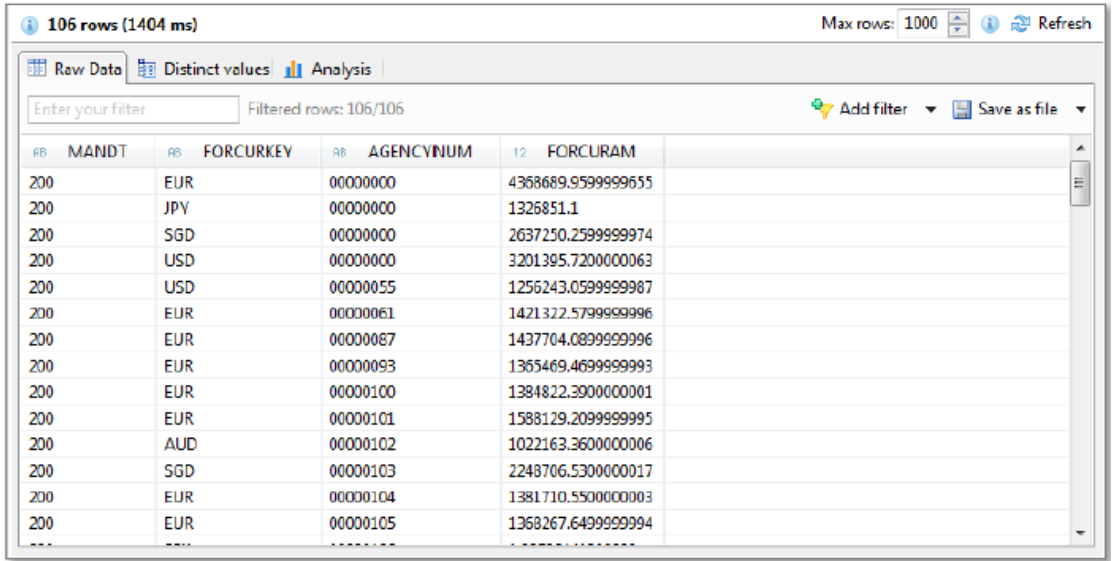
在Modeler的底部，我们可以看到两个标签，Data Foundation和Logical View。在右侧有一个滑杆来改变显示模型的尺寸大小。尝试操作滑块，熟悉它的行为。单击Logical View，有个小矩形标为Data Foundation，它的属性列表是空的。切换回Data Foundation标签，右击属性MANDT使之加到逻辑视图中。在弹出框的菜单中选择Add as Attribute。

请添加属性 FURCURKEY（外币的数字键）和 AGENCYNUM（旅行编号）到输出表中，和你添加 MANDT 方式一样。

右击属性 FURCURKEY（外币总和），然后选择 *Add as Measure*，查看输出表和逻辑视图验证你的结果。

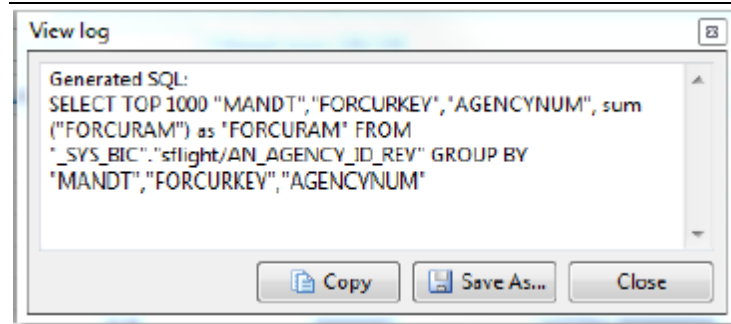
保存你的模型通过从下拉框菜单中选择 *File ► Save*，或者你也可以按 *Ctrl-S*。你新建的分析视图会在导航栏界面中作为包 *sflight* 节点可见，它的图标标有一个灰色的钻石，这代表该模型已经保存在元库中，但是还没有被激活，没有运行时对象生成在 *_SYS_BIC* 模式下。通过右击导航栏中的 *AN_AGENCY_ID_REV* 节点，选择 *Activate* 来激活分析视图。灰色钻石消失表明激活成功。

恭喜！你刚刚已经建立了你的第一个SAP HANA分析视图，但是能运行吗？右击 *AN_AGENCY_ID_REV*，然后选择 *Data Preview*。在主界面中，你会看到刚刚创建的分析视图中的数据以表格形式显示。



#B	MANDT	#B	FORCURKEY	#B	AGENCYNUM	12	FORCURAM
200			EUR		00000000		4358689.95999999655
200			JPY		00000000		1326851.1
200			SGD		00000000		2637250.2599999974
200			USD		00000000		3201395.7200000063
200			USD		00000055		1256243.0599999987
200			EUR		00000061		1421322.5799999996
200			EUR		00000067		1437704.0899999996
200			EUR		00000093		1365469.4699999993
200			EUR		00000100		1384822.3900000001
200			EUR		00000101		1588129.2099999995
200			AUD		00000102		1022163.3600000006
200			SGD		00000103		2248706.5300000017
200			EUR		00000104		1381710.5500000003
200			EUR		00000105		1368267.6499999994

点击界面右上角的信息图标显示用于检索数据的SQL语句。现在我们用一分钟时间解析下这条语句。为了可读性，我们省去了所有多余的引号。



```
SELECT TOP 1000 MANDT, FORCURKEY, AGENCYNUM, SUM(FORCURAM) AS FORCURAM
FROM _SYS_BIC."sflight/AN_AGENCY_ID_REV"
GROUP BY MANDT, FORCURKEY, AGENCYNUM
```

该语句从位于模式 _SYS_BIC（存放所有生成的运行时对象）下的视图 sflight/AN_AGENCY_ID_REV 取得数据，属性 MANDT、FORCURKEY、和 AGENCYNUM 被选中。除此之外，列 FORCURAM 作为该属性计算和显示，在 MANDT, FORCURKEY 和 AGENCYNUM 上进行分组。

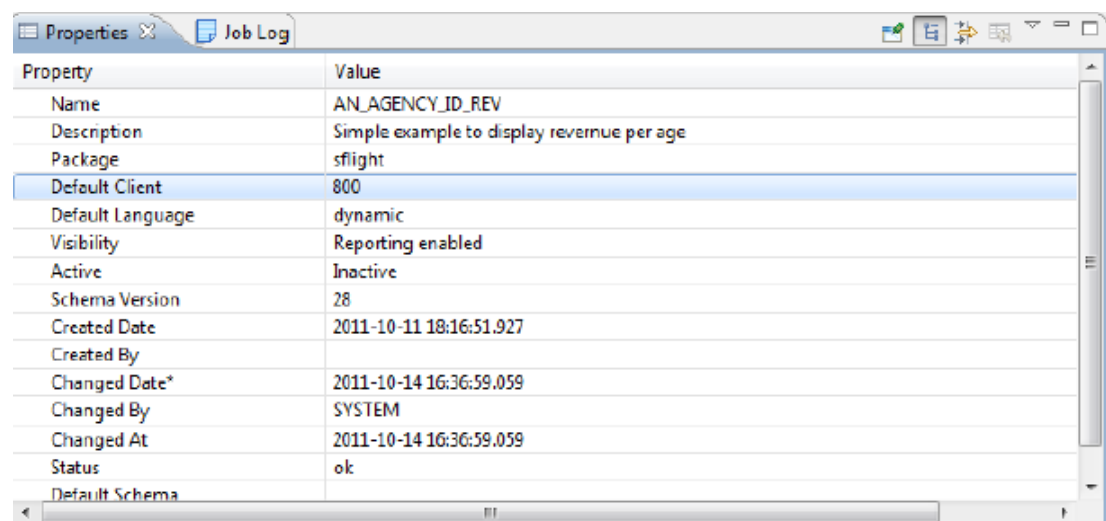
那么，数据预览（data preview）显示什么内容？对于每一个旅行社（更具体一点，数字 ID），根据货币显示收入。我们可以知道编号为 00000000 的旅行社收入由欧元（欧洲）、日元（日本）和新加坡元（新加坡）构成。

3.4 MANDT 属性

我们表显示了所有 MANDT 属性可能的值的条目，这是没有用的，因为不同的 MANDT 值区分了在一个 SAP ERP 系统中不同的公司，例如控股公司及其下属公司，把不同公司的收入加起来是没有意义的。

我们的数据集包含了 MANDT 两个值 200 和 800 的数据，因此，我们只从后者中选择数据。在 Modeler 的下方，还有一个 SAP HANA studio 界面显示了分析视图的属性。属性 *Default Client* 默认值是 *dynamic*，双击值域可以进行编辑，请把值设为 800。

保存分析视图并激活。数据预览现在只显示了 MANDT 为 800 的行。



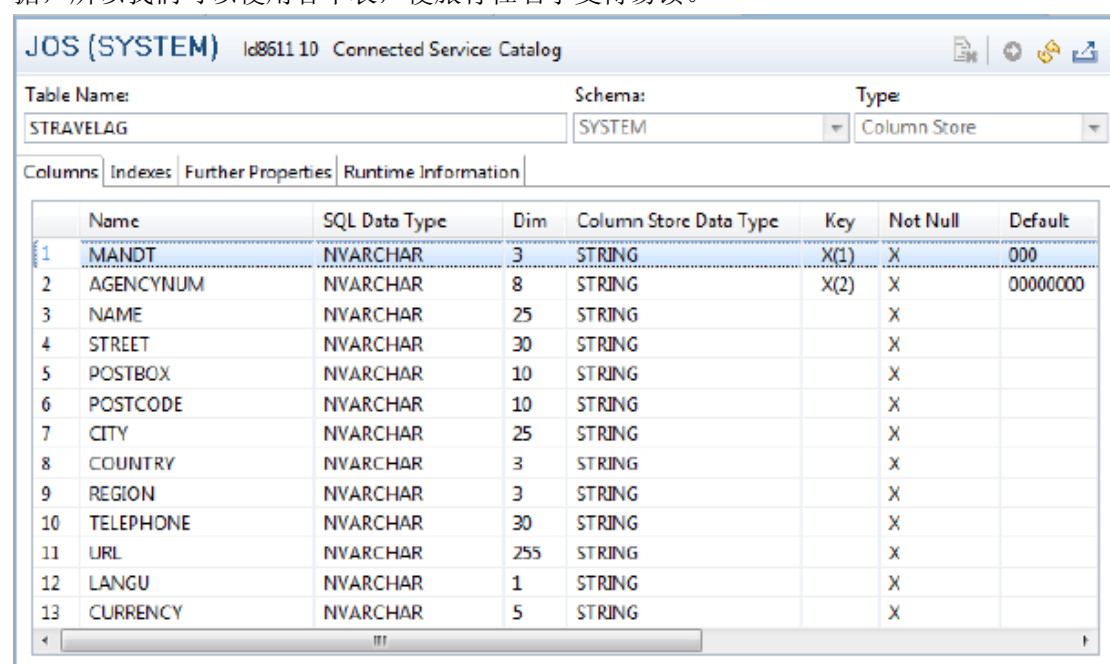
3.2.5 应用过滤

让我们先检查欧洲的收入，这可以通过在属性 **FORCURKEY** 上设置过滤实现。过滤是可以设置在分析视图和属性视图中。

在 *Data Foundation* 标签中，右击 **FORCURKEY** 选择 *Apply Filter*。弹出的对话框可以让你选择一些操作符，例如 **Between** 或是 **Equal**。请选择 **Equal**，输入值 **EUR**。保存你的模型并激活。现在数据预览只显示了 **FORCURKEY** 为 **EUR** 的行。

3.2.6 创建属性视图

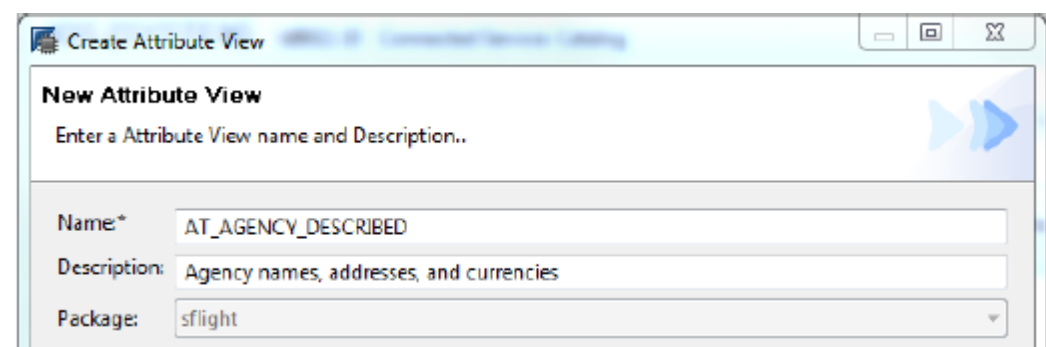
现在回到我们的问题：每个旅行社制造了多少收入？如果我们能记住所有旅行社的 **ID**，我们就可以利用已经有的数据。另一方面，有一张 **STRAVELAG** 表里面包含了每个旅行社的数据，所以我们可以使用者个表，使旅行社名字变得易读。



	Name	SQL Data Type	Dim	Column Store Data Type	Key	Not Null	Default
1	MANDT	NVARCHAR	3	STRING	X(1)	X	000
2	AGENCYNUM	NVARCHAR	8	STRING	X(2)	X	00000000
3	NAME	NVARCHAR	25	STRING		X	
4	STREET	NVARCHAR	30	STRING		X	
5	POSTBOX	NVARCHAR	10	STRING		X	
6	POSTCODE	NVARCHAR	10	STRING		X	
7	CITY	NVARCHAR	25	STRING		X	
8	COUNTRY	NVARCHAR	3	STRING		X	
9	REGION	NVARCHAR	3	STRING		X	
10	TELEPHONE	NVARCHAR	30	STRING		X	
11	URL	NVARCHAR	255	STRING		X	
12	LANGU	NVARCHAR	1	STRING		X	
13	CURRENCY	NVARCHAR	5	STRING		X	

这是一张主数据表，所以我们来建一个分析视图只选择我们感兴趣的属性，比如 **NAME**, **CITY**, **COUNTRY**, **CURRENCY**。

在导航栏界面中，打开包 *sflight* 右击 **Attribute Views** 节点，选择 **New ► Attribute View**。在弹出的对话框中输入一个名字，例如 **AT_AGENCY_DESCRIBED** 和描述，点击 **FINISH**。



Create Attribute View

New Attribute View

Enter a Attribute View name and Description..

Name*:

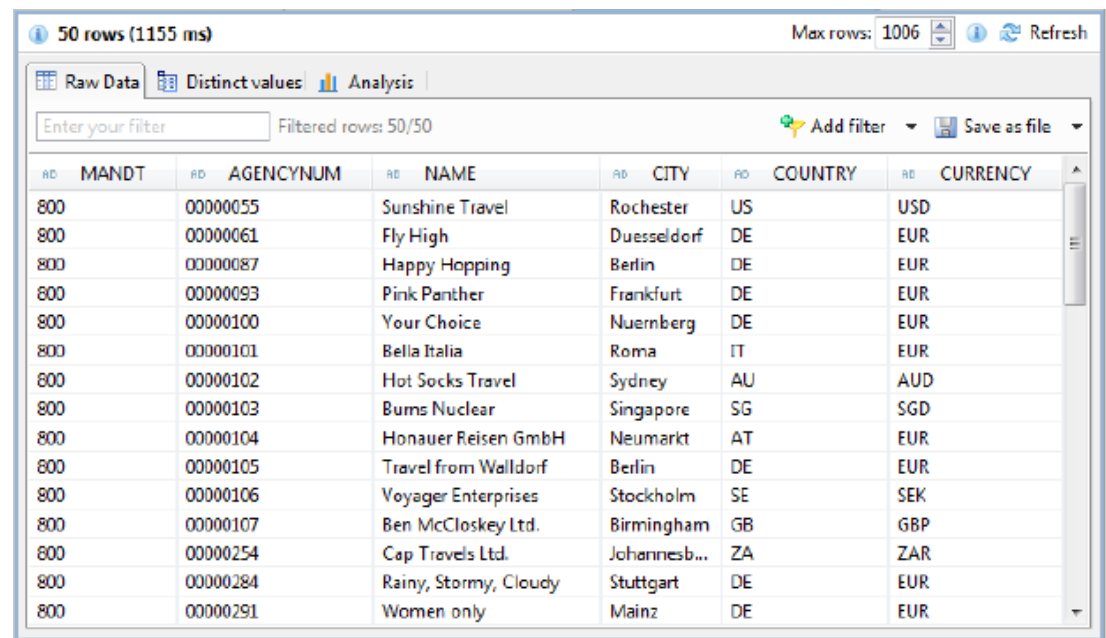
Description:

Package:

在导航栏界面中，浏览到 *Catalog / SYSTEM / Tables*，单击表节点 **STRAVELAG** 并拖拽至 *Modeler* 中的 *sflight.AT_AGENCY_DESCRIBED* 画板。请把 **MANDT**, **NAME**, **CITY**, **COUNTRY** 和 **CURRENCY**

定义成视图的属性，添加 **AGENCYNUM** 为主属性。

为了不混淆不同公司的旅行社，修改视图属性 *Default Client* 从 *dynamic* 改为 **800**。请保存新的属性视图并激活。现在在导航栏界面中有个新的节点 **Content / sflight / AT_AGENCY_DESCRIBED**，右击节点，然后选择 **Data Preview** 查看编号为 **800** 的所有旅行社清单。



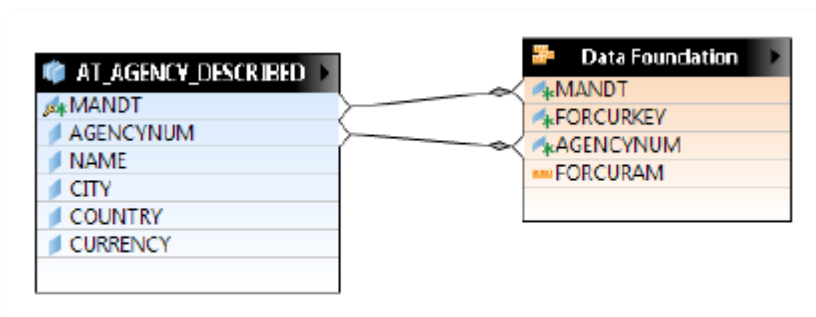
ID	MANDT	ID	AGENCYNUM	ID	NAME	ID	CITY	ID	COUNTRY	ID	CURRENCY
800		00000055			Sunshine Travel		Rochester		US		USD
800		00000061			Fly High		Duesseldorf		DE		EUR
800		00000087			Happy Hopping		Berlin		DE		EUR
800		00000093			Pink Panther		Frankfurt		DE		EUR
800		00000100			Your Choice		Nuernberg		DE		EUR
800		00000101			Bella Italia		Roma		IT		EUR
800		00000102			Hot Socks Travel		Sydney		AU		AUD
800		00000103			Burns Nuclear		Singapore		SG		SGD
800		00000104			Honauer Reisen GmbH		Neumarkt		AT		EUR
800		00000105			Travel from Walldorf		Berlin		DE		EUR
800		00000106			Voyager Enterprises		Stockholm		SE		SEK
800		00000107			Ben McCloskey Ltd.		Birmingham		GB		GBP
800		00000254			Cap Travels Ltd.		Johannesb...		ZA		ZAR
800		00000284			Rainy, Stormy, Cloudy		Stuttgart		DE		EUR
800		00000291			Women only		Mainz		DE		EUR

3.2.7 构建星型架构

我们现在已经准备好把属性视图和分析视图联接成一个简单的星型架构，我们将会用属性视图作为架构的一个“手臂”，表 **SBOOK** 作为中心。我们将从上节中新建的分析视图来构建星型架构。

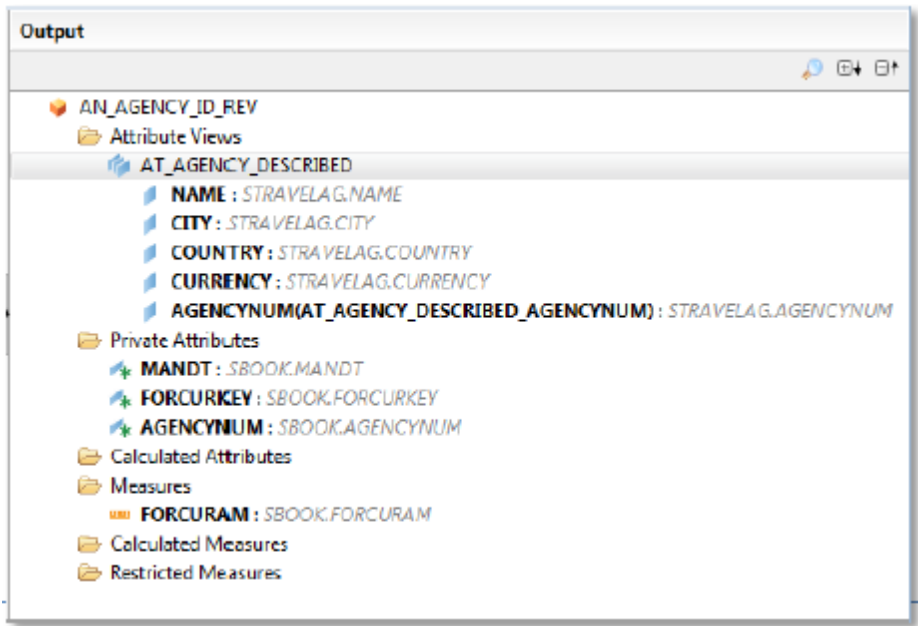
请在包 **sflight** 下创建一个新的分析视图，名为 **AN_AGENCY_REV**。在创建对话框中点击 *Copy From*，选择 **sflight / Analytic Views / AN_AGENCY_ID_REV**。

切换到 **Modeler** 画板下的 **Logical View** 标签，拖拽属性视图 **AT_AGENCY_DESCRIBED** 至画板上。



不用对警告消息感到惊讶。属性 **AGENCYNUM** 在两个表之间有一个名称冲突。SAP HANA Modeler 为新建的属性创建一个别名来避免冲突。

单击在 **Data Foundation** 中的属性 **AGENCYNUM**，然后拖拽至新建的属性视图中的 **AGENCYNUM**。以同样的方式连接 **MANDT**。这样就创建了属性 **MANDT** 和 **AGENCYNUM** 上



打开位于 Modeler 画板右侧输出结构中的节点 *Attribute Views / AT_AGENCY_DESCRIBED*。你将看到我们的属性视图已经和 SBOOK 中的属性相连，最初用来创建分析视图 AN_AGENCY_ID_REV 和之后的 AN_AGENCY_REV。

The screenshot shows the 'Output' window displaying a table with 22 rows. The table has the following columns: NAME, CITY, COUNTRY, CURRENCY, AT_AGENCY_DESCRIBED_AGENCYNUM, MANDT, FORCURKEY, and FORCURAM. The data is as follows:

NAME	CITY	COUNTRY	CURRENCY	AT_AGENCY_DESCRIBED_AGENCYNUM	MANDT	FORCURKEY	FORCURAM
Pink Panther	Frankfurt	DE	EUR	00000093	800	EUR	1364241.729999999
Your Choice	Nuernberg	DE	EUR	00000100	800	EUR	1355369.510000000
Travel from Walldorf	Berlin	DE	EUR	00000105	800	EUR	1353477.35
Bella Italia	Roma	IT	EUR	00000101	800	EUR	1588944.270000000
Honsauer Reisen GmbH	Neumarkt	AT	EUR	00000104	800	EUR	1369901.110000000
Caribbean Dreams	Emden	DE	EUR	00000117	800	EUR	1367463.880000000
Everywhere	Muenchen	DE	EUR	00000119	800	EUR	1377253.879999999
Special Agency Peru	Stuttgart	DE	EUR	00000116	800	EUR	1368918.839999999
Bavarian Castle	Dresden	DE	EUR	00000110	800	EUR	1359611.200000000
Happy Holiday	Bremen	DE	EUR	00000120	800	EUR	1374914.709999999
Maxitrip	Wiesbaden	DE	EUR	00000294	800	EUR	1368047.559999999
Fly Low	Dresden	DE	EUR	00000122	800	EUR	1597717.329999999
No Return	Koeln	DE	EUR	00000115	800	EUR	1349118.029999999
Up 'n' Away	Hannover	DE	EUR	00000124	800	EUR	1353964.209999999
Fly & Smile	Frankfurt	DE	EUR	00000188	800	EUR	1090183.280000000
Rainy, Stormy, Cloudy	Stuttgart	DE	EUR	00000284	800	EUR	1618936.119999999

我们新的分析视图可以用来展示每个旅行社的欧元收入，形式比我们第一个分析视图只能用来查看旅行社编号而不是名字等，更容易理解。
请检查下新建的分析视图的属性。默认的客户端应该设成 800，以便于只有这个客户端的数据才可以结合到星型模式中。

3.2.8 计算属性

在我们接下去的例子中，我们将会在分析视图中做些计算。我们要像之前一样的方式计算收入，但是，现在我们要在欧元以 10% 的折扣支付所有预订的假设下进行计算。

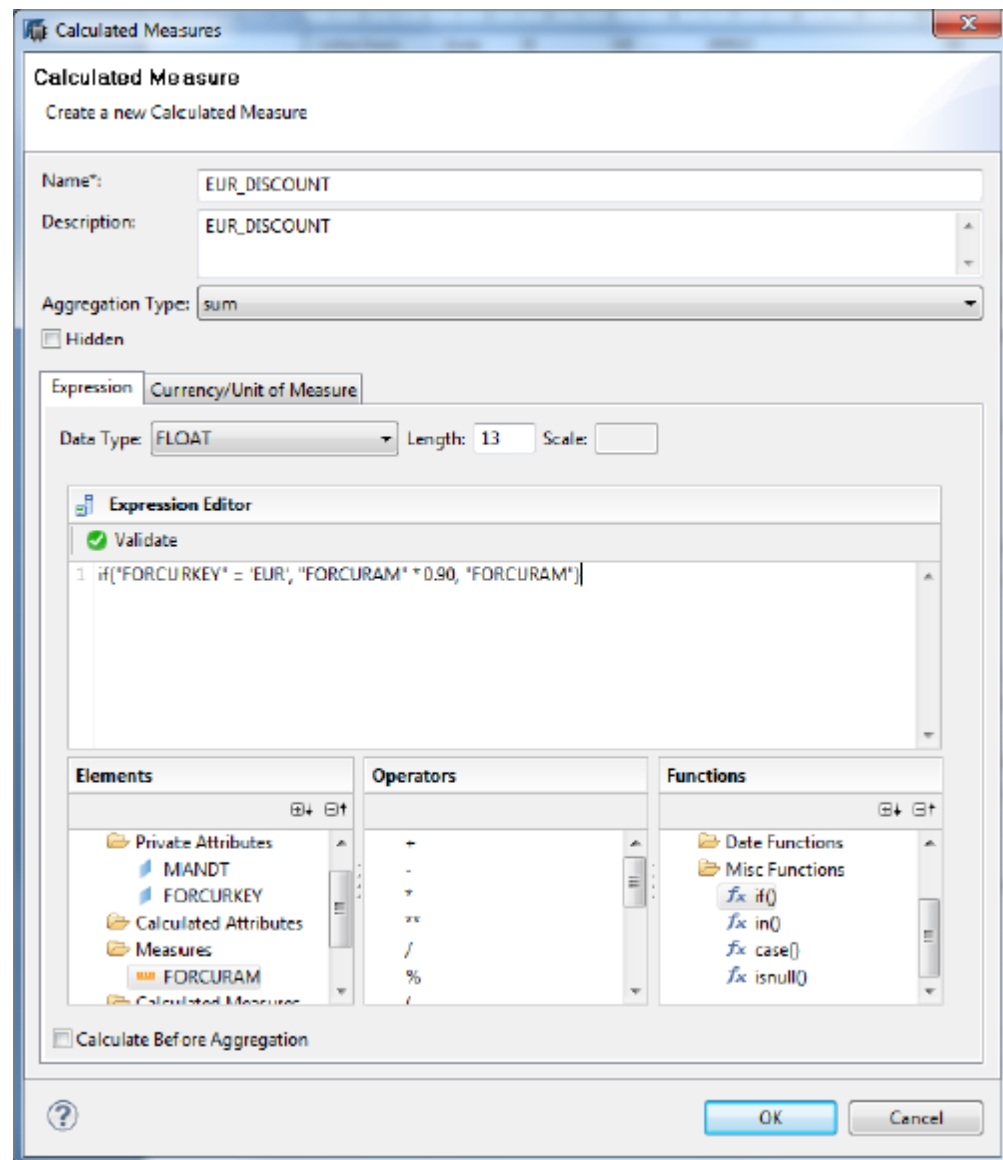
请在包 *sflight* 下新建一个分析视图名为 *AN_AGENCY_EUR_DISCOUNT*，在弹出的对话框中点击 *Copy From* 然后选择 *sflight / Analytic Views / AN_AGENCY_REV*。

记得在我们上个例子中，我们把结果限定在收入为欧元的范围内，现在我们要去除这一限制。右击在 **Modeler** 窗格中的 **FORCURKEY** 属性，选择 *Remove Filter*。

在 **Modeler** 画板右侧的 **Output** 窗口中，请右击 *Calculated Measures* 选择 *New*。在对话框中输入如下 Name = **EUR_DISCOUNT**, Data Type = **FLOAT**。在表达式编辑器中，输入 `if("FORCURKEY" = 'EUR', "FORCURAM" * 0.90, "FORCURAM")`。这相当于：如果属性 **FORCURKEY** 等于字符‘EUR’，那么 **FORCURAM** 应当乘以 0.9 返回，否则就应该使用 **FORCURAM** 原始值。

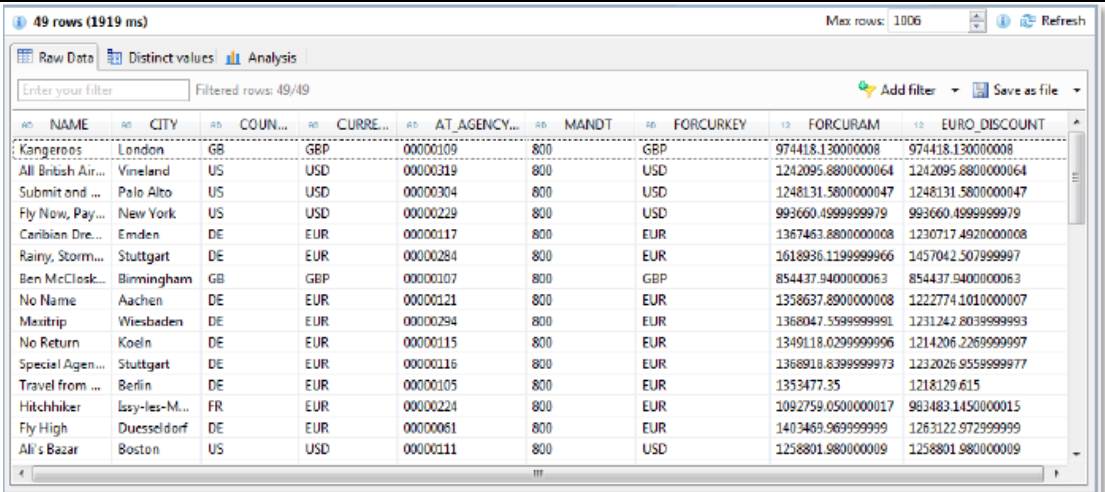
手动输入的表达式也可以通过拖拽下方的编辑器窗口菜单中的表达式元素进行组装。

点击 **OK**，保存表达式。



请保存并激活分析视图。

现在，数据预览根据我们的计算表达式显示了额外的一列，名为 **EUR_DISCOUNT**。



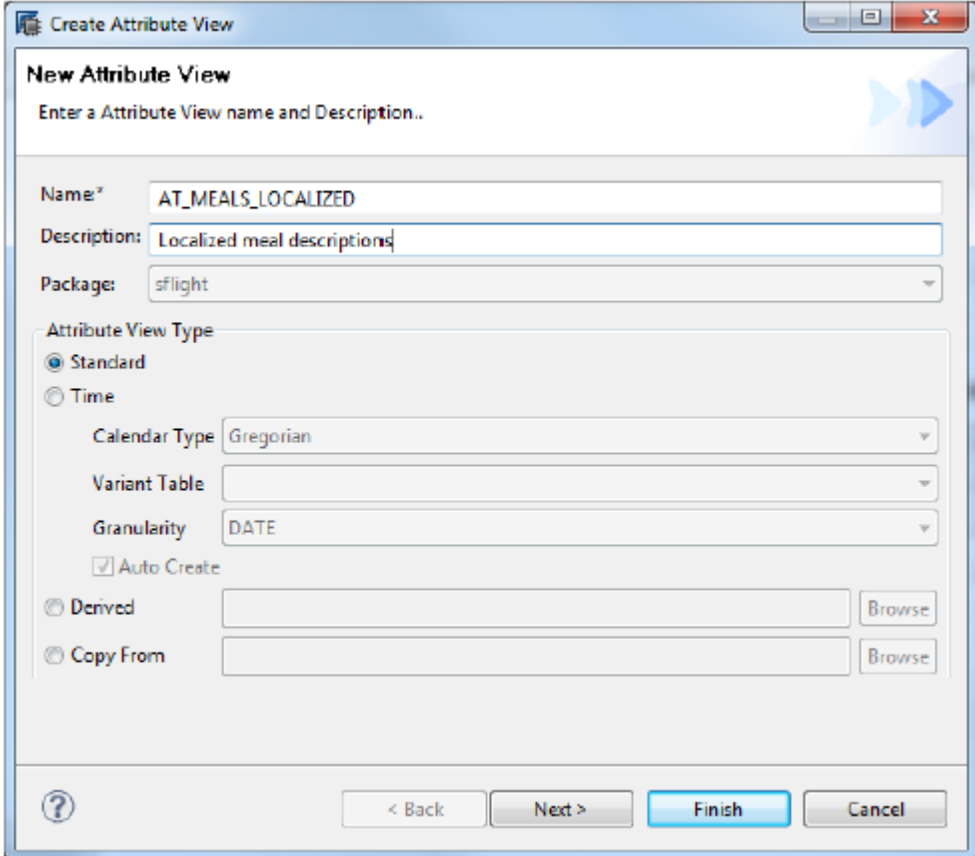
NAME	CITY	COUNTRY	CURRENCY	AT_AGENCY	MANDT	FORCURKEY	FORCURAM	EURO_DISCOUNT
Kangeroos	London	GB	GBP	00000109	800	GBP	974418.130000008	974418.130000008
All British Air...	Vineland	US	USD	00000319	800	USD	1242095.8800000064	1242095.8800000064
Submit and ...	Palo Alto	US	USD	00000304	800	USD	1248131.5800000047	1248131.5800000047
Fly Now, Pay...	New York	US	USD	00000229	800	USD	993660.4999999979	993660.4999999979
Caribbean Dre...	Emden	DE	EUR	00000117	800	EUR	1367463.8800000008	1230717.4920000008
Rainy, Storm...	Stuttgart	DE	EUR	00000284	800	EUR	1618936.1199999966	1457042.507999997
Ben McClosk...	Birmingham	GB	GBP	00000107	800	GBP	854437.9400000063	854437.9400000063
No Name	Aachen	DE	EUR	00000121	800	EUR	1358637.8900000008	1222774.1010000007
Mexitrip	Wiesbaden	DE	EUR	00000294	800	EUR	1368047.5599999991	1231242.8039999993
No Return	Koeln	DE	EUR	00000115	800	EUR	1349118.0299999996	1214206.2260999997
Special Agen...	Stuttgart	DE	EUR	00000116	800	EUR	1368918.8399999973	1232026.9559999977
Travel from ...	Berlin	DE	EUR	00000105	800	EUR	1353477.35	1218129.615
Hitchhiker	Iasy-les-M...	FR	EUR	00000224	800	EUR	1092759.0500000017	983483.1450000015
Fly High	Duesseldorf	DE	EUR	00000061	800	EUR	1403469.9699999999	1263122.9729999999
Ali's Bazar	Boston	US	USD	00000111	800	USD	1258801.980000009	1258801.980000009

3.2.9 多语言属性视图（文本视图）

属性使用用来展示事实表更详细的数据。往往这些细节只有以用户的语言展示才能被理解。在本节中，我们将根据用户区域，从多语言的主数据表中自动选择文本。

考虑一张物料预定的事实表，其中的物料只能由物料号提供。你可以从物料主数据表中创建一个属性视图，然后和事实表相联接，构成一个星型架构，如同我们从分析视图 AN_AGENCY_REV 中的表 SBOOK 和 STRAVELAG 创建的那样。

在 SFLIGHT 有两个表，可以和属视图相结合，提供依赖于语言的描述文本的选择。在接下去的存储过程概述了创建依赖语言的属性视图的步骤，基于语言的属性视图称为文本视图。请打开在导航栏界面中的包节点 Content / sflight, 右击 *Attribute Views* 选择 **New ► Attribute View**。在对话框中输入名字 AT_MEALS_TRANSLATED，单击 Next。



The 'Create Attribute View' dialog box is shown. The 'New Attribute View' section has the following fields:

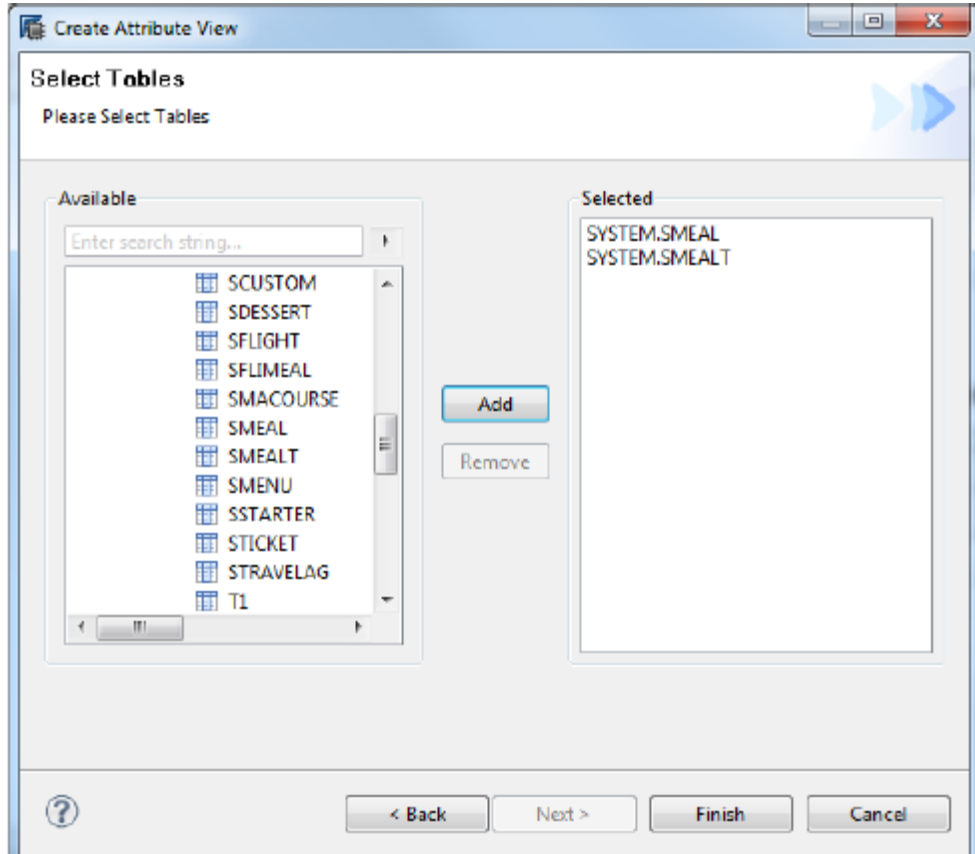
- Name: AT_MEALS_LOCALIZED
- Description: Localized meal descriptions
- Package: sflight

The 'Attribute View Type' section has the following options:

- ☒ Standard
- ☐ Time
 - Calendar Type: Gregorian
 - Variant Table:
 - Granularity: DATE
 - ☒ Auto Create
- ☐ Derived
- ☐ Copy From

At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

从可选的列表中选择表 SMEAL 和 SMEALT。勾选它们，单击 *Add* 按钮把它们加到已选择表的列表中，单击 Finish。

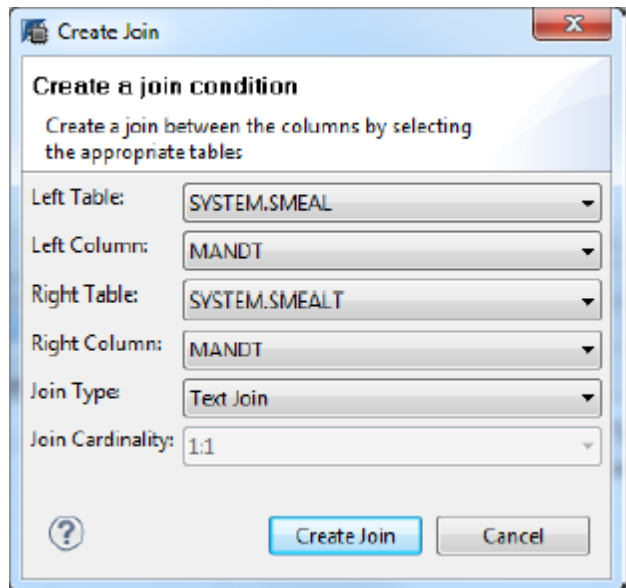


The 'Select Tables' dialog box is shown. It has two main sections:

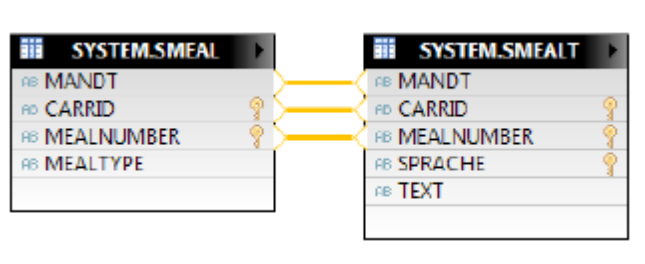
- Available:** A list of tables with checkboxes. The tables listed are SCUSTOM, SDESSERT, SFLIGHT, SFLIMEAL, SMACOURSE, SMEAL, SMEALT, SMENU, SSTARTER, STICKET, STRAVELAG, and T1. The 'Add' button is located to the right of this list.
- Selected:** A list of tables that have been added. The tables listed are SYSTEM:SMEAL and SYSTEM:SMEALT. The 'Remove' button is located to the left of this list.

At the bottom, there are buttons for '< Back', 'Next >', 'Finish', and 'Cancel'.

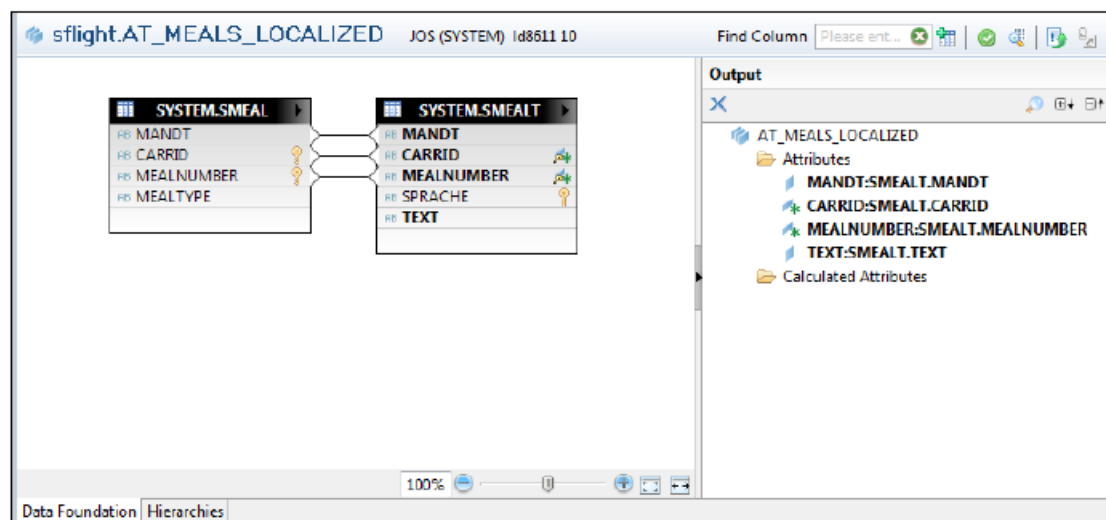
在 *Data Foundation* 标签中右击 *Modeler* 画板, 选择 *Create Join*。创建一个在 *SMEAL* 和 *SMEALT* 之间的 *MANDT* 属性的文本联接, 如下图所示。现在, 单击 *Create Join*。



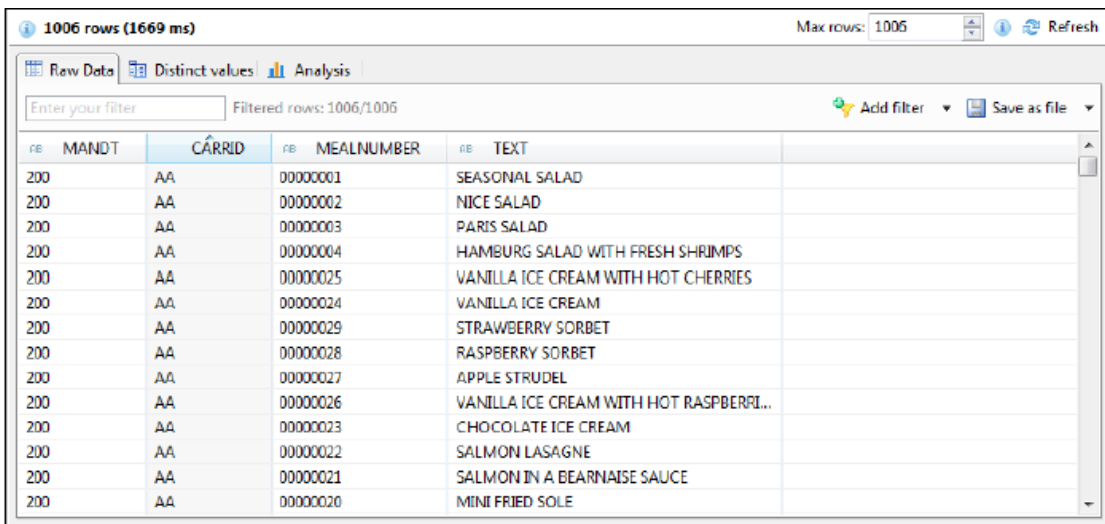
同样, 再创建两个属性 *CARRID* 和 *MEALNUMBER* 之间的联接。



下一步, 添加 *MANDT* 到输出结构的列表中, 添加 *CARRID* 和 *MEALNUMBER* 为主属性, 新增 *TEXT* 到输出结构中。



保存模型并激活视图。从我们的新的文本视图导航栏中调用 *Data Preview* 功能, 如果您单击列 *CARRID* 进行排序, 你会看到所有用餐的名字出现在您的默认语言。



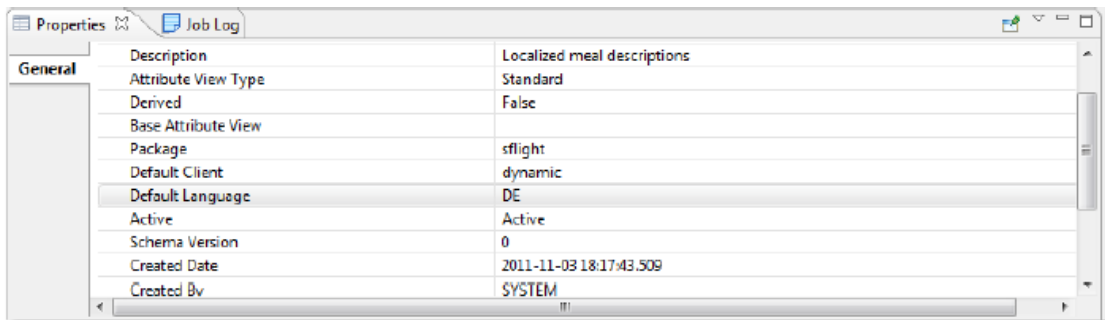
1006 rows (1669 ms) Max rows: 1006 Refresh

Raw Data Distinct values Analysis

Enter your filter Filtered rows: 1006/1006 Add filter Save as file

MANDT	CARRID	MEALNUMBER	TEXT
200	AA	00000001	SEASONAL SALAD
200	AA	00000002	NICE SALAD
200	AA	00000003	PARIS SALAD
200	AA	00000004	HAMBURG SALAD WITH FRESH SHRIMPS
200	AA	00000025	VANILLA ICE CREAM WITH HOT CHERRIES
200	AA	00000024	VANILLA ICE CREAM
200	AA	00000029	STRAWBERRY SORBET
200	AA	00000028	RASPBERRY SORBET
200	AA	00000027	APPLE STRUDEL
200	AA	00000026	VANILLA ICE CREAM WITH HOT RASPBERRI..
200	AA	00000023	CHOCOLATE ICE CREAM
200	AA	00000022	SALMON LASAGNE
200	AA	00000021	SALMON IN A BEARNAISE SAUCE
200	AA	00000020	MINI FRIED SOLE

要显示除了默认语言以外的用餐描述，可以通过按照 3.2.2 节列出的存储过程，给视图属性 *Default Language* 设置不同的值。

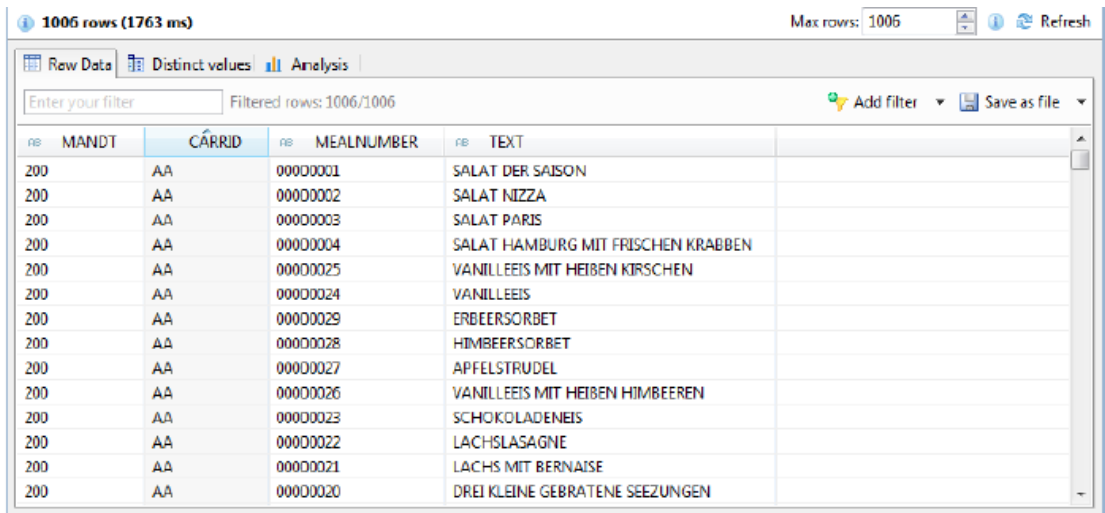


Properties Job Log

General

Description	Localized meal descriptions
Attribute View Type	Standard
Derived	False
Base Attribute View	
Package	sflight
Default Client	dynamic
Default Language	DE
Active	Active
Schema Version	0
Created Date	2011-11-03 18:17:43.509
Created By	SYSTEM

保存视图并激活。现在数据预览以所选的语言显示所有用餐描述。请注意，数据包 *SFLIGHT* 没有包含所有语言的完整描述，但是，英语和德语除外。



1006 rows (1763 ms) Max rows: 1006 Refresh

Raw Data Distinct values Analysis

Enter your filter Filtered rows: 1006/1006 Add filter Save as file

MANDT	CARRID	MEALNUMBER	TEXT
200	AA	00000001	SALAT DER SAISON
200	AA	00000002	SALAT NIZZA
200	AA	00000003	SALAT PARIS
200	AA	00000004	SALAT HAMBURG MIT FRISCHEN KRABBen
200	AA	00000025	VANILLEIS MIT HEISSEN KIRSCHEN
200	AA	00000024	VANILLEIS
200	AA	00000029	ERBEERSORBET
200	AA	00000028	HIMBEERSORBET
200	AA	00000027	APFELSTRUDEL
200	AA	00000026	VANILLEIS MIT HEISSEN HIMBEEREN
200	AA	00000023	SCHOKOLADENEIS
200	AA	00000022	LACHSLASAGNE
200	AA	00000021	LACHS MIT BERNAISE
200	AA	00000020	DREI KLEINE GEBRATENE SEEZUNGEN

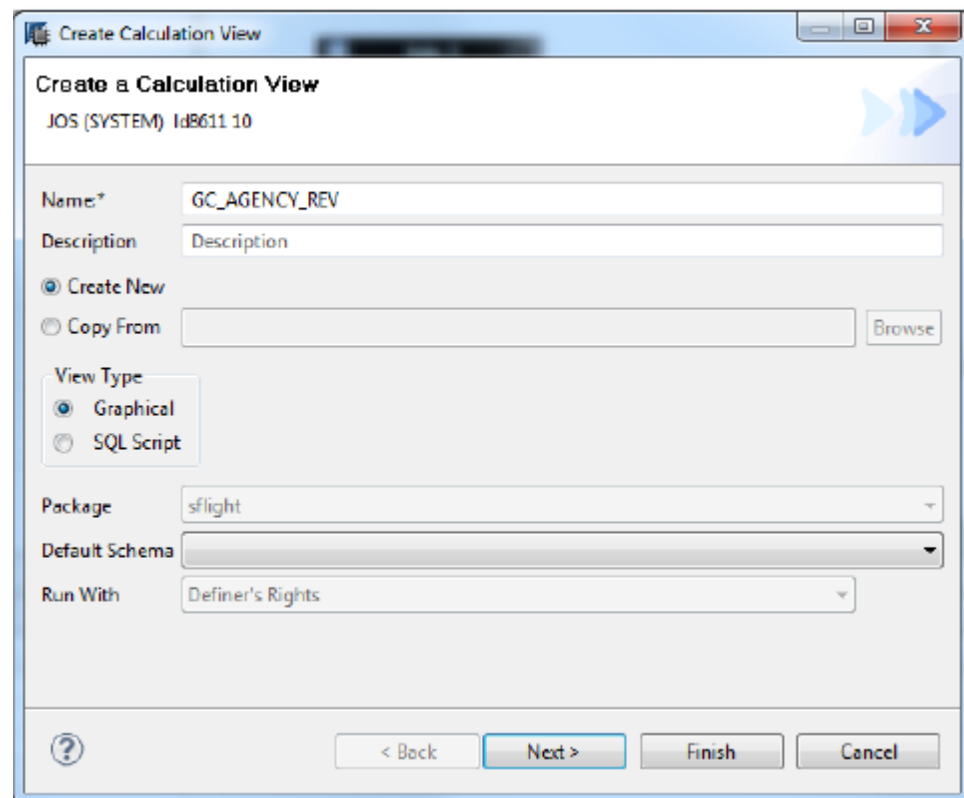
请注意，在模型视图中常用的默认的客户端属性在文本视图中不适用。单击 *Add filter* 然后选择在 *Column filters* 窗格中选择 *MANDT = 800*，显示只来自该客户的数据。

3.2.10 图形化计算视图

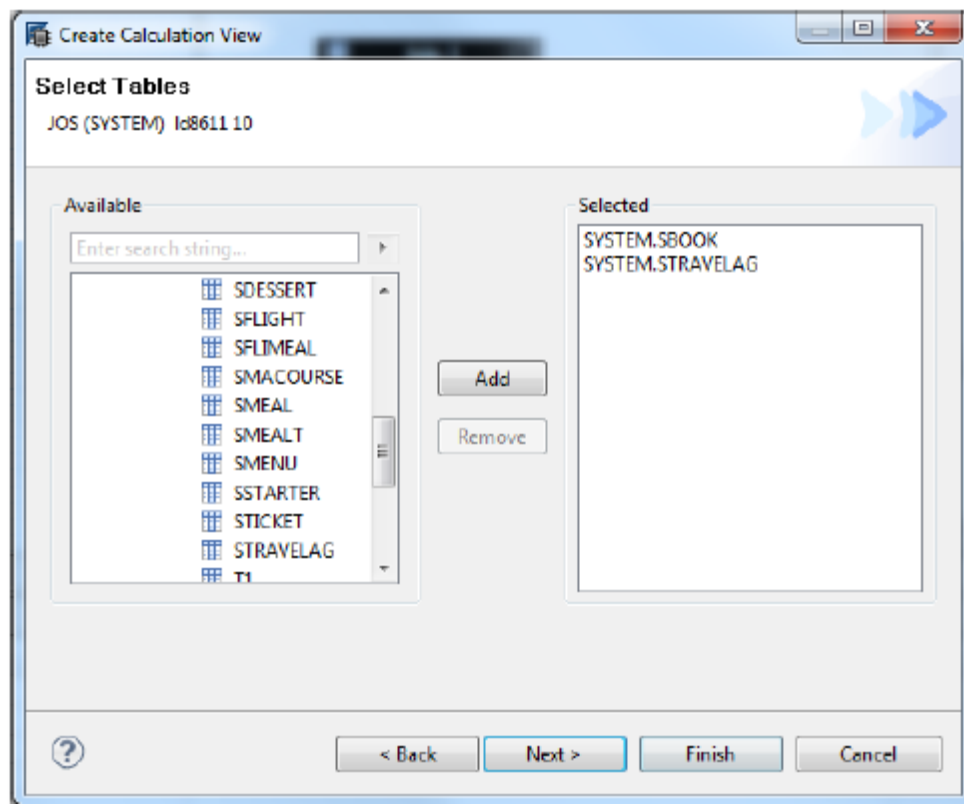
在 SAP HANA 数据库有几种方式创建视图。我们刚学习了属性视图和分析视图，以及把它们

结合形成星型架构，本身也是一个分析视图。这些视图完全都是由列存储来处理。对于更复杂的视图，我们需要利用内置的函数，这些视图称为计算视图。

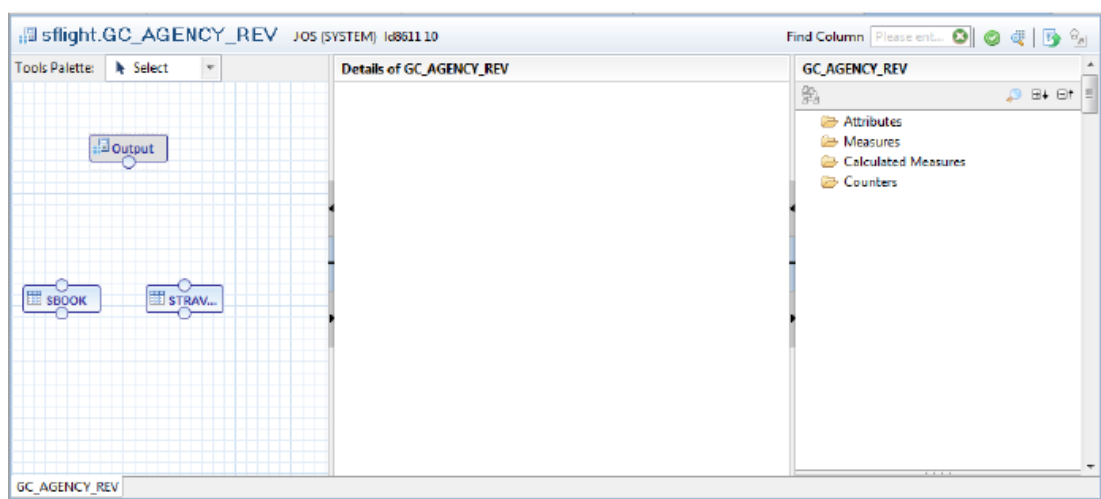
作为我们下个例子，我们将全部重建例子 AN_AGENCY_REV，作为图形计算视图。在导航栏界面中打开包节点 *Content / sflight*。右击 *Calculation Views*，选择 *New ► Calculation View*，在对话框中输入视图名字 GC_AGENCY_REV，单击 Next。



从可选表的列表中选择 *SBOOK* 和 *STRAVELAG*，通过勾选它们单击 *Add* 按钮，把它们加到选中表列表中，点击 *Finish*。



Modeler 画板将出现，并分成两块区域。左边的是用来连接表与输出节点，运用从 *Tools Palette* 选择的各種操作。当图形计算视图建模程序第一次启动时，节点 *Output* 会被选中。选中的节点来决定右边画板区域的显示内容。

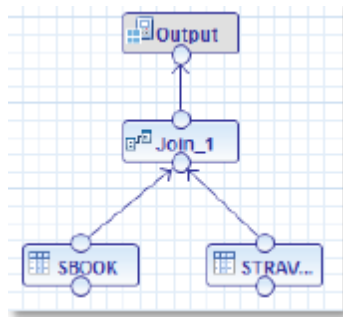


在左侧区域我们可以看到有三个盒子或节点名为 SBOOK, STRAVELAG 和 Output。它们表示图形视图使用的两张表和输出。为了对之前的例子 AN_AGENCY_REV 重新建模，我们将联接两张表并且把结果属性和节点 output 相连。

让我们从 tools palette 中添加联接操作。请选择 Join，把新的节点 Join_1 移到 SBOOK, STRAVELAG 和 Output 的中间。

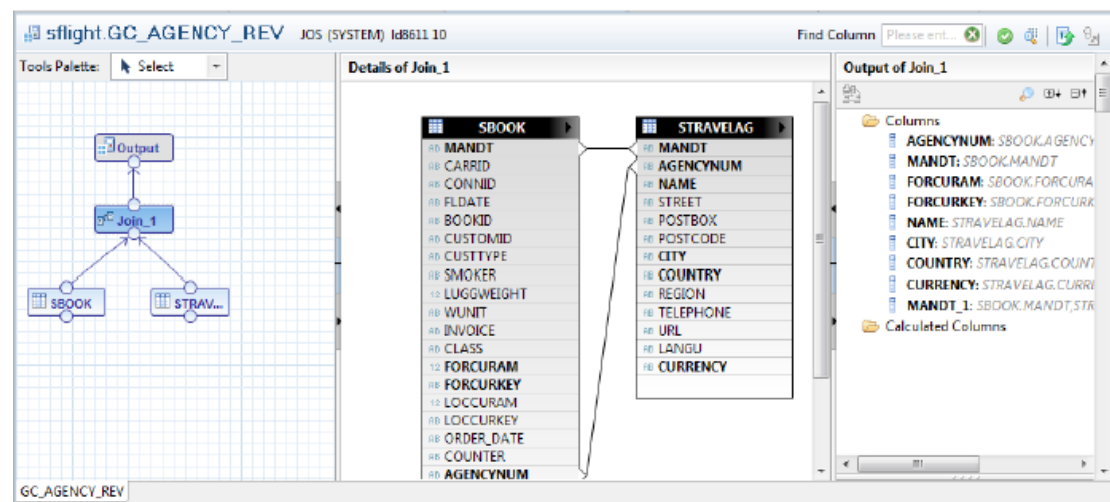
现在以如下方式连接四个节点：单击 SBOOK 顶端的圆圈，然后拖出一条直线到 Join_1 顶端

的圆圈来连接这两个节点。以右图所示的方式连接其他节点。

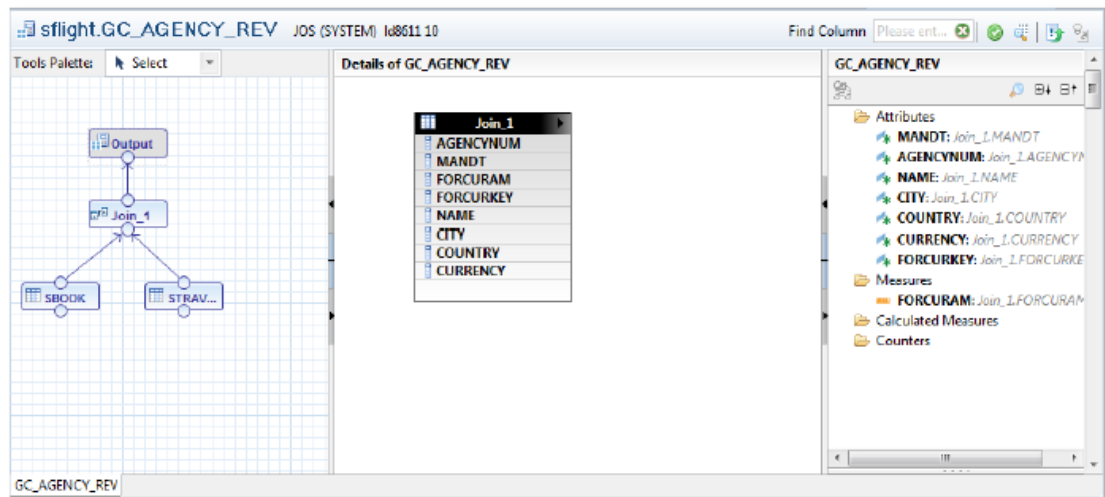


现在，单击 **Join_1** 展示画板中间区域的 join 的详细信息。你将会看到两张表 **SBOOK** 和 **STRAVELAG** 的属性，这两张表需要以共有字段 **AGENCYNUM** 进行联接。单击 **SBOOK** 表的 **AGENCYNUM**，然后画出到 **STRAVELAG** 表的 **AGENCYNUM** 的连接。

右击 **SBOOK** 表中的 **MANDT**，然后添加到联接的 **Output** 中，对 **SBOOK** 表的 **FORCURAM** 和 **FORCURKEY** 以及 **STRAVELAG** 表的 **NAME**, **CITY**, **COUNTRY** 以及 **CURRENCY** 执行同样的操作。



我们刚才定义了联接操作导出的列或属性，接下去，我们要把它们分配到节点 **Output** 或是结果节点导出的属性类型。单击画板左侧区域的 **Output**，在中间的位置，请选择 **MANDT**, **AGENCYNUM**, **NAME**, **CITY**, **COUNTRY**, **CURRENCY** 和 **FORCURKEY** 作为属性，**FORCURAM** 为度量。保存计算视图并激活。



请注意，在模型视图中常用的默认的客户端属性在计算视图中不适用。为了比较分析视图 AN_AGENCY_REV 和我们新视图 CG_AGENCY_REV 的结果，我们需要在数据预览中，使用各自的过滤。单击 *Add filter*，在 *Column filters* 窗格中选择 MANDT=800。

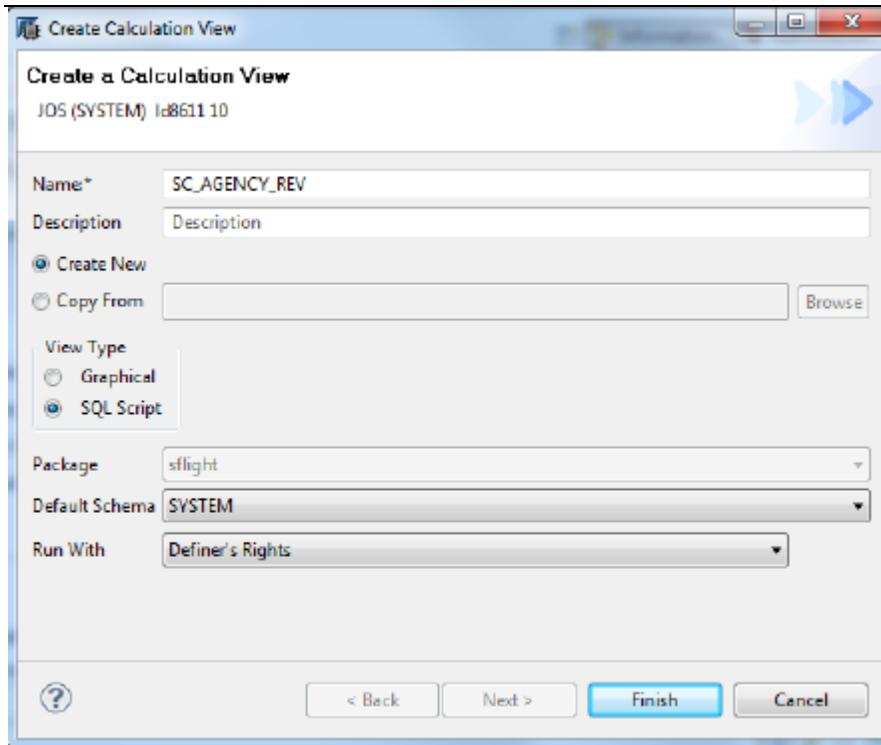
The screenshot displays the SAP HANA Studio interface for the 'GC_AGENCY_REV' view. The 'Raw Data' tab is selected, showing a table with 98 rows (1280 ms) and 1006 max rows. The table is filtered to 49 rows where MANDT=800. The columns are: MANDT, AGENCYNUM, NAME, CITY, COUNTRY, CURRENCY, FORCURAM, and FORCURKEY. The 'Column filters' pane on the right shows the filter 'MANDT (1/2) - Equal' with a value of 800.

MANDT	AGENCYNUM	NAME	CITY	COUNTRY	CURRENCY	FORCURAM	FORCURKEY
800	00000055	Sunshine Travel	Rochester	US	USD	USD	1243959,20
800	00000061	Fly High	Duesseldorf	DE	EUR	EUR	1403469,97
800	00000087	Happy Hopping	Berlin	DE	EUR	EUR	1414573,75
800	00000093	Pink Panther	Frankfurt	DE	EUR	EUR	1364241,73
800	00000100	Your Choice	Nuernberg	DE	EUR	EUR	1355369,51
800	00000101	Bella Italia	Roma	IT	EUR	EUR	1588944,27
800	00000102	Hot Socks Travel	Sydney	AU	AUD	AUD	1027026,94
800	00000103	Burns Nuclear	Singapore	SG	SGD	SGD	2252937,74
800	00000104	Honauer Reisen GmbH	Neumarkt	AT	EUR	EUR	1369901,11
800	00000105	Travel from Walldorf	Berlin	DE	EUR	EUR	1353477,35
800	00000106	Voyager Enterprises	Stockholm	SE	SEK	SEK	13218578,39
800	00000107	Ben McCloskey Ltd.	Birmingham	GB	GBP	GBP	854437,94
800	00000108	Pillepalle Trips	Zuerich	CH	CHF	CHF	2019691,05
800	00000109	Kangeroos	London	GB	GBP	GBP	974418,13
800	00000110	Bavarian Castle	Dresden	DE	EUR	EUR	1359611,20

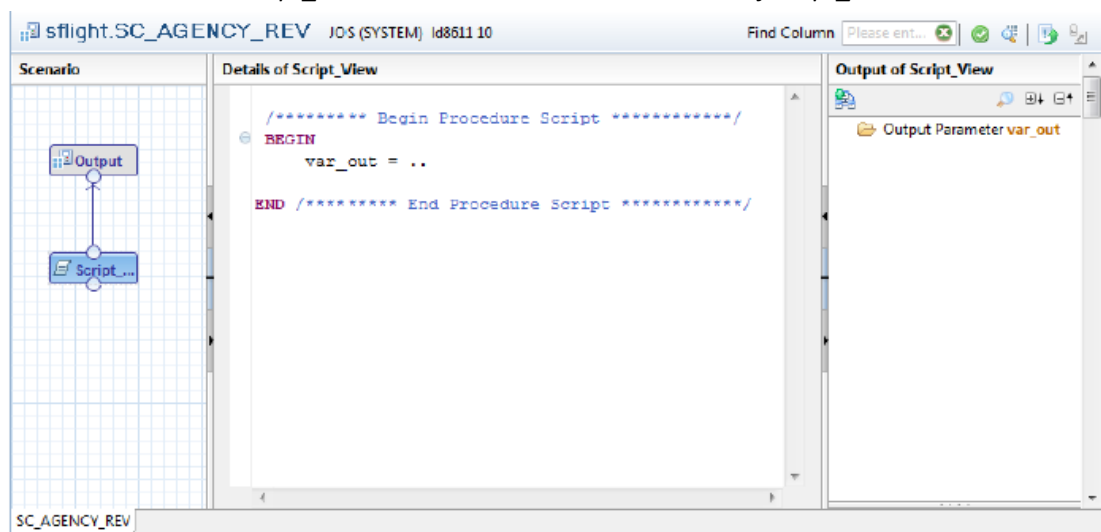
3.2.11 如何创建脚本计算视图

脚本计算视图是我们这个教程中讨论的模型视图的最后一类。例子和之前的很类似，我们会创建分析视图 AN_AGENCY_REV 另一个实现方式，这次使用的是 SQLScript。有关 SQLScript 的详细信息，请阅读 [SAP HANA SQLScript documentation](#)。

在导航栏界面中打开包节点 *Content / sflight*，右击 *Calculation Views*，然后选择 *New* ► *Calculation View*，在对话框中请输入视图名字 *SC_AGENCY_REV*，*View Type* 选择 *SQLScript*，点击 *Finish*。



现在点击模型节点 *Script_View*，打开脚本编辑器窗口 *Details of Script_View*。








编辑器显示一个 SQLScript 代码片段，其中需要你完成。请在编辑器窗口中的 **BEGIN** 和 **END** 之间输入如下内容：

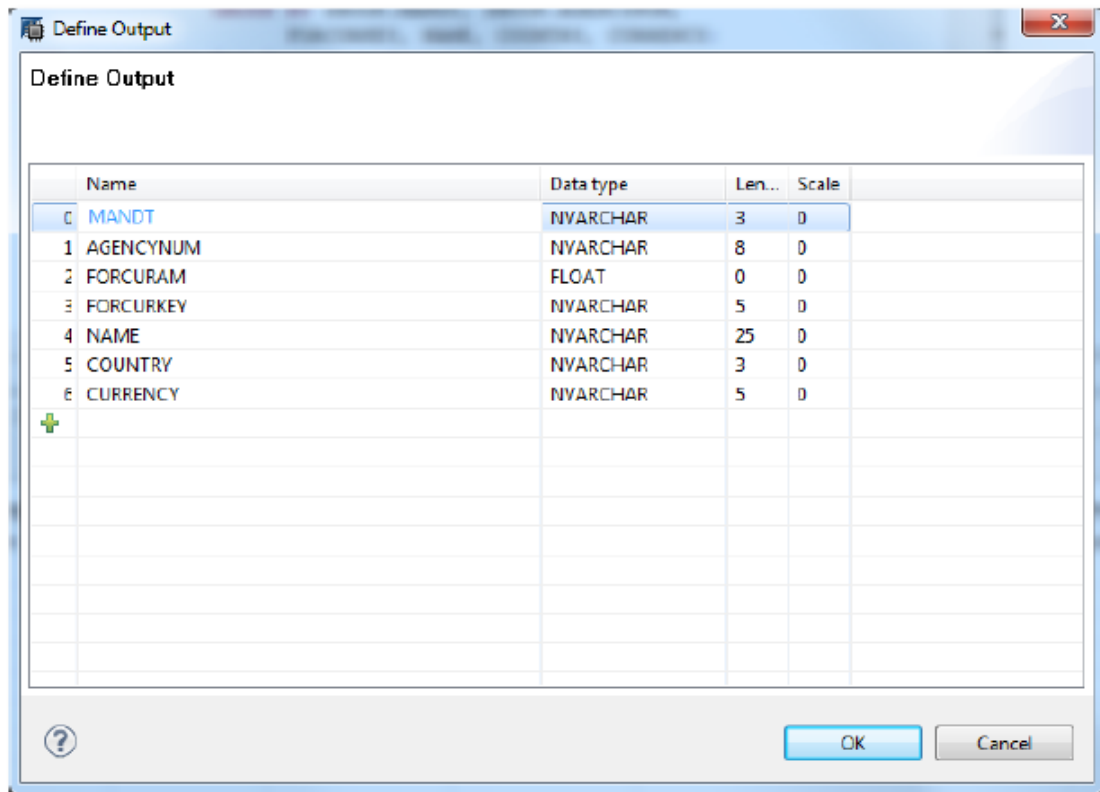
```
var_out = SELECT SBOOK.MANDT, SBOOK.AGENCYNUM,  
SUM(SBOOK.FORCURAM) as FORCURAM,  
FORCURKEY, NAME, COUNTRY, CURRENCY  
FROM SYSTEM.SBOOK, SYSTEM.STRAVELAG  
WHERE SBOOK.AGENCYNUM = STRAVELAG.AGENCYNUM  
AND SBOOK.MANDT = STRAVELAG.MANDT  
GROUP BY SBOOK.MANDT, SBOOK.AGENCYNUM,  
FORCURKEY, NAME, COUNTRY, CURRENCY;
```

这条语句选出的数据与视图 *AN_AGENCY_REV* 和 *GC_AGENCY_REV* 中的完全一样。

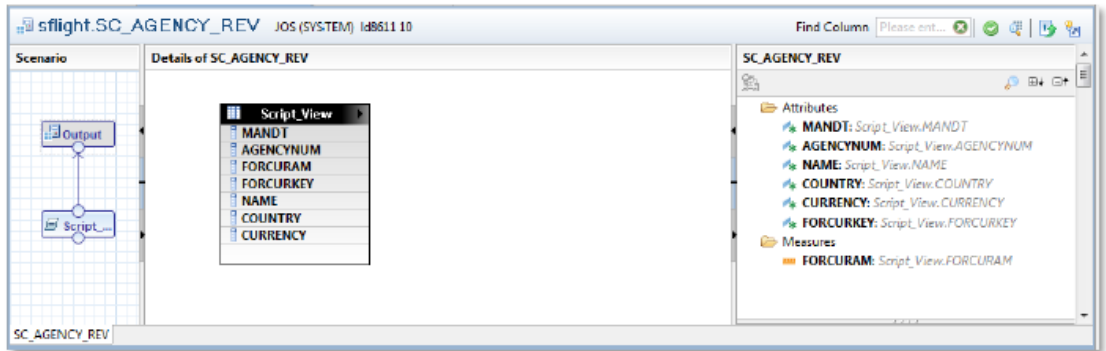
在返回变量 `var_out` 中可见的属性有

- SBOOK.MANDT,
- SBOOK.AGENCYNUM,
-  FORCURAM (sum aggregate),
-  SBOOK.FORCURKEY,
-  STRAVELAG.NAME,
-  STRAVELAG.COUNTRY,
-  STRAVELAG CURRENCY

这些属性必须手动声明为脚本视图节点的输出属性。请点击 *Output of Script View* 窗口中的图标，然后输入属性以及根据下图的数据类型。单击绿色加号来添加另一行，点击 **ok** 保存设置。



由于 Script View 节点会导出这些属性,我们需要把它们分配到 Output 节点中。请选择 MANDT, AGENCYNUM, NAME, COUNTRY, CURRENCY 和 FORCURKEY 作为属性, FORCURAM 为度量。保存视图并激活。



请再注意，在模型视图中常用的默认的客户属性在计算视图中不适用。在数据预览中，单击 **Add filter**，在 **Column filters** 窗格中选择 **MANDT=800**。

MANDT	AGENCYNUM	NAME	COUNTRY	CURRENCY	FORCURKEY	FORCURAM
800	00000055	Sunshine Travel	US	USD	USD	1243959,20
800	00000061	Fly High	DE	EUR	EUR	1403469,97
800	00000087	Happy Hopping	DE	EUR	EUR	1414573,75
800	00000093	Pink Panther	DE	EUR	EUR	1364241,73
800	00000100	Your Choice	DE	EUR	EUR	1355369,51
800	00000101	Bella Italia	IT	EUR	EUR	1588944,27
800	00000102	Hot Socks Travel	AU	AUD	AUD	1027026,94
800	00000103	Burns Nuclear	SG	SGD	SGD	2252937,74
800	00000104	Honauer Reisen G...	AT	EUR	EUR	1369901,11
800	00000105	Travel from Weildorf	DE	EUR	EUR	1353477,35
800	00000106	Voyager Enterprises	SE	SEK	SEK	13218578,39
800	00000107	Ben McCloskey Ltd.	GB	GBP	GBP	854437,94
800	00000108	Pillepalle Trips	CH	CHF	CHF	2019691,05
800	00000109	Kangeros	GB	GBP	GBP	974418,13
800	00000110	Bevarian Castle	DE	EUR	EUR	1359611,20
800	00000111	Ali's Bazar	US	USD	USD	1258801,98
800	00000112	Super Agency	GB	GBP	GBP	851754,52
800	00000113	Wang Chong	RU	USD	USD	1252591,94

分析和属性视图局限于简单的数据模型，例如，不能联接两张表作为数据基础。在这种情况下，脚本视图可以作为解决方案，因为其可以用 **SQLScript** 执行复杂的计算。请了解计算的复杂程度决定了 **SAP HANA** 的响应时间。在第 4 章，我们将讨论不同的方法，对 **SAP HANA** 进行性能调优。

3.3 使用 SAP HANA studio 执行 SQL 和 SQLscript 语句

许多其他的数据库系统以形成物理存储的数据库区分彼此，目录和模式仅仅作为命名空间（目录包含模式）。通常，一个私人的模式以每个用户账号创建，**SAP HANA** 把模式存在分开的数据库中。这不是什么问题，没有必要去区分不同的物理存储，因为系统中所有的信息都放在内存中。

一个 **SAP HANA** 设备由一个数据库组成，反过来，一个数据库有一个目录组成。

模式是不同的命名空间，例如，同样的表名或视图名会出现在不同的模式中。可以授权在模式，表或视图的级别上，授予的权限不会自动传播到模式中已存在的其他对象。

如果已经为你新建一个用户，你现在应该能看到一个和你名字相同的模式，如果你连接使用的是 **SYSTEM** 用户，情况就并非如此了。请右击各自的模式节点（你的用户名或者 **SYSTEM**），然后在上下文菜单中选择 **SQL 编辑器（SQL Editor）**。这样就打开了一个新的界面，你可以在里面输入 **SQL** 语句然后传送给 **SAP HANA** 数据库。请展开在导航栏界面中各自的节点，来查看模式中已有的对象。

你现在将创建 **SAP HANA** 数据库中的第一张表。请在 **SQL 编辑器** 界面中输入如下 **SQL** 语句，

或者写在一行中或者像显示的那样分几行：

```
create table local_climate (  
location nvarchar(60),  
temperature_low int,  
temperature_high int,  
annual_precipitation float );
```

执行该语句，请单击文本区域上面带箭头的绿色圆圈或按 F8 键。你现在应该能看到一条成功消息显示在 SQL 文本区域的下方。这条消息也告诉你 SQL 语句的执行时间。你可以看到很容易就能计算出执行时间，这对于调优您的数据库架构和查询是很有必要的。如果你能访问本文档的在线版本，你可以用操作系统的剪切和粘贴功能来复制 SQL 语句到 SQL 编辑器中。

现在，我们将用一些真实的数据来填充表。请执行如下的 SQL 语句。你可以输入多条语句，由分号分隔（“;”），并用鼠标点击一个绿色圆圈执行。

```
insert into local_climate values ('Berlin', -2, 24, 570.7);  
insert into local_climate values ('Heidelberg', 3, 20, 745.0);  
insert into local_climate values ('Biel', -2, 25, 1203.0);  
insert into local_climate values ('Dublin', 5, 15, 769.0);  
insert into local_climate values ('Milano', -2, 29, 943.2);  
insert into local_climate values ('Vancouver', -1, 22, 1181.5);
```

为了检查表中的内容，请执行：

```
select * from local_climate;
```

你现在应该在 SQL 编辑器中看到第二个名为“Result (1)”的标签，其中显示了你输入的数据。列 ANNUAL_PRECIP 中的 Berlin 和 Milano 的数据显示了很多小数点位数。这是由于一般情况下，实际值不能以无限精度存储。在这些情况下，SAP HANA 会选择最精确的内部表示方法。请点击导航栏界面并按 F5，刷新列表。然后请单击你选择的模式中 Tables 节点前的小三角，你现在应该看到名为 LOCAL_CLIMATE 的新节点，这表示你新建的表。双击树形节点，这会在同一屏幕区域带出和 SQL Editor 一样的界面。它显示了 SAP HANA 对于该表所知道的内容：列名和类型，以及自动选择的内部表类型。从这我们了解到，SQL 数据类型和 SAP HANA 内部允许高效的内存存储的数据类型相匹配。在表格显示的上方，有三个字段标记 Table Name, Schema 和 Type，从后者我们知道，表 LOCAL_CLIMATE 以默认的行存储方式创建。让我们删除该表并以列存储方式重建表，执行如下语句：

```
drop table local_climate;  
create column table local_climate (  
location nvarchar(60),  
temperature_low int,  
temperature_high int,  
annual_precipitation float  
);
```

重复上面的步骤来填充数据到表中，并显示其结构。你可以在结构视图中看到，现在该表的表类型是列式存储。请删除该表，以行存储方式重建表，并用如前所述的数据填充。我们需

要这张行存储的表来进行进一步的说明。

你或许注意到在导航栏界面中，新的表名为 LOCAL_CLIMATE，而在 SQL 语句中使用的是“local_climate”。SQL 对 SQL 关键字和未加引号的名字不区分大小写。如果你想使用大小写混合的名称，请在这些名字加双引号，例如"LOCAL_CLIMATE"和"Local_Climate"就是不同的名字。SQLScript 是 SAP HANA 内置的 SQL 扩展语言，用来实现快速内存计算。请执行下面的语句创建一个新的表名为 PRECIPITATION，和表 CLIMATE 中的两列具有相同的名称和类型。表 PRECIPITATION 将会在稍后的章节中作为临时表的模板。

```
create table precipitation (  
location nvarchar(60),  
annual_precipitation real );
```

下一条执行的语句是真正的 SQLScript 代码，它声明了一个存储过程，将在后面的代码被调用(**call ...**)。存储过程有两个输入参数 LOW_TEMP_MIN 和 LOW_TEMP_MAX，以及一个输出参数 PRCPT。输入参数的类型都是整数型，而输出参数的结构和表 PRECIPITATION 一样。该过程选择表 LOCAL_CLIMATE 的两列 LOCATION 和 ANNUAL_PRECIP，只选择那些 LOW_TEMP 的值介于区间[LOW_TEMP_MIN ... LOW_TEMP_MAX]的行，并把查询的结果放到输出参数 PRCPT。

```
create procedure "getColdPrecipitation" (  
in low_temp_min integer,  
in low_temp_max integer,  
out prcpt precipitation  
)  
language SQLScript reads SQL data as  
begin  
prcpt = select location, annual_precipitation from local_climate  
where temperature_low between :low_temp_min AND :low_temp_max;  
end;
```

请执行如下的语句，其以表 PRECIPITATION 为模板创建了一张临时表名为 PRC。

```
create global temporary table prc like precipitation;
```

现在我们调用之前定义的过程来填充表 PRC:

```
call "getColdPrecipitation"(0, 5, prc) with overview;
```

你会注意到有一个新的标签出现在 SQL 编辑器界面中，单击标签可以看到 getColdPrecipitation 返回的内容。

SQLScript 存储过程返回了一张表，其中，第一列是本地输出参数，第二列为分配的值或是表。

现在在同样的 SQL 编辑器中，执行语句：

```
select * from prc;
```

注意，我们使用的是同一个 SQL 编辑器，这样可以利用同样的数据库连接，否则临时表将会为空。从显示的结果你可以看到表 CLIMATE 中被选出的行的列 LOW_TEMP 值都为 0 到 5。最后一个在 SAP HANA Studio 中使用 SQL 编辑器的快速说明，正如前文所述，你可能一次输入所有的语句到编辑器中，然后点击绿色圆圈或是按 F8 来执行。如果你只标记文本中的一部分点击绿色圆圈，只有被标记的文本会被 SAP HANA 数据库执行。

欲了解更多有关 SQLScript 的内容，请参阅 *SAP HANA Database SQLScript* 文档。

3.4 如何显示查询计划

SQL 语句的查询计划解释了执行语句所必须进行的基本操作，例如扫描表、使用索引、应用过滤和联接的计算。在接下去的章节中，我们将利用表 CLIMATE 和 PRC 的联接来展示 SAP HANA 的查询计划功能。

请执行如下语句：

```
Select * from local_climate, prc where local_climate.location = prc.location;
```

这条语句选择和显示了表 CLIMATE 所有的行，其中列 LOCATION 的值也出现表 PRC 相应的列中。点击名为"Result (1)"的标签显示了查询结果，包含了地方为 Heidelberg 和 Dublin 的两行。请在 SQL 编辑器中标记上述语句，右击文本，并选择"Explain Plan"。这会重新创建"Result (1)"标签，含有执行的操作符和详细信息的列表。基本上是反向显示 SAP HANA 的 SQL 处理器调用的堆栈。这个例子还不是很有启发性。让我们通过应用一个过滤到选择语句来重建这些步骤。请执行并解释如下语句：

```
select local_climate.* from local_climate, prc where local_climate.location = prc.location and prc.location = 'Dublin';
```

执行之后，"Result (1)"标签显示了地方为 Dublin 的一行记录。这些数据的计算方式由查询计划来解释。右击该语句，执行"Explain Plan"会把很多数据填充至标签"Result (1)"。

列出的第一个操作是基于几条记录或对于字段 LOCAL_CLIMATE.LOCATION, LOCAL_CLIMATE.LOW_TEMP, LOCAL_CLIMATE.TEMPERATURE_HIGH, LOCAL_CLIMATE.ANNUAL_PRECIPITATION 的行搜索。它们是上文中的"select local_climate.*"语句所需要的。这步操作组装了元组，用来显示或是由列表中的下一个操作计算的消费。

接下来的操作是哈希联接。从这两张表选中的行用哈希函数来联接，比较 LOCATION 的值。利用哈希值而非原始字符串，用时可以减少，尤其对大数据集，因此，这种比较方式被选为默认的方式。

第三个操作是对于表 CLIMATE 的扫描，来找到所有地方为“Dublin”的行。因为表 LOCATION 没有创建索引，所以整表扫描是必须的。因此，表 CLIMATE 中的每条记录都要和过滤条件"LOCAL_CLIMATE.LOCATION = 'Dublin'"进行核对。为什么表 LOCAL_CLIMATE 要检查地方为 Dublin 的记录，我们就不用指定表 PRC 也进行检查吗？如果你看了下面的两行，你会发现表 PRC 也被检查了。SAP HANA 的语句优化器发现为了有效的计算联接，过滤条件必须同时应用到两张表中。

第四个操作是对列 PRC.LOCATION 的搜索。这里我们注意到和第三个操作的不同之处。由于 LOCAL_CLIMATE 以行存储方式重建，PRC 在内存中列式存储，使用不同的操作来从两张表中选择数据。

如果你滚动操作表至右侧的话，你会发现在列 PARENT_OPERATOR_ID 中，操作三和四都作为操作二的一部分执行，计算联接。这是合理的，因为联接需要结合不同表中的数据。

现在让我们在列 LOCAL_CLIMATE.LOCATION 上新建一个索引，请执行

```
create index climate_loc on local_climate (location);
```

然后执行并解释语句：


```
select local_climate.* from local_climate, prc where local_climate.location = prc.location and  
prc.location = 'Dublin';
```

现在，如果你查看查询计划，你会看到操作二和操作三被相关模式双聚类（CPB）的树形索引联接所取代。

总体而言，对行存储引擎的表使用索引是个好主意，因为其可以精确地选择数据，这在计算大表的联接时特别重要，这样减少了两表中需要联接的元组，加快了计算速度。有一个折中的索引可以进行更快的查询，但插入数据时会有一个开销，因为需要维护索引。

3.4.1 查询计划中的列

下表介绍了在查询计划中显示的列。

Column Name	Description
STATEMENT_NAME	对STATEMENT_NAME指定的字符串执行EXPLAIN PLAN命令。这是用来区分彼此的计划，当有多个计划在EXPLAIN_PLAN_TABLE视图。
OPERATOR_NAME	操作符名，在下节中详细描述
OPERATOR_DETAILS	操作符的详细信息，谓词和操作使用的表达式都在这里显示。
SCHEMA_NAME	访问表所在的模式名。
TABLE_NAME	访问表名。
TABLE_TYPE	访问表类型。在下列选项之一： COLUMN TABLE, ROW TABLE, MONITORING VIEW, JOIN VIEW, OLAP VIEW, CALCULATION VIEW and HIERARCHY VIEW。
TABLE_SIZE	访问表中行的估计数目。
OUTPUT_SIZE	操作符生成行的估计数目。
SUBTREE_COST	从操作符开始执行子树的估计代价。该值只是为了相对比较。
OPERATOR_ID	计划中唯一操作标识符ID，从1开始计数。
PARENT_OPERATOR_ID	OPERATOR_ID双亲的标识符ID。SQL计划的形状是一棵树，树的拓扑结构可以使用OPERATOR_ID和PARENT_OPERATOR_ID重建。根操作符的PARENT_OPERATOR_ID显示为NULL。
LEVEL	根操作符的等级。根的等级为1，其子操作等级为2，等等。这可用以输出缩进。
POSITION	双亲操作符的位置。第一个子操作符的位置为1，第二个为2，等等。
HOST	执行操作符所在的主机名。
PORT	用来连接主机的TCP/IP端口。
TIMESTAMP	EXPLAIN PLAN命令执行的日期和时间。
CONNECTION_ID	执行EXPLAIN PLAN的连接ID。
EXECUTION_ENGINE	执行操作符的引擎类型：行或列。

3.4.2 执行计划中的 OPERATOR_NAME 列

下面是查询计划中在 OPERATOR_NAME 显示的列引擎操作符列表。

COLUMN SEARCH	列引擎操作符的起始位置。OPERATOR_DETAILS 列出了投影的列。
LIMIT	限制输出行数的操作符。
ORDER BY	对输出行排序的操作符。
HAVING	过滤顶端谓词分组和聚集的操作符。
GROUP BY	分组和聚集的操作符。
DISTINCT	消除重复的操作符。
FILTER	用谓词进行过滤的操作符。
JOIN	联接输入关系的操作符。
COLUMN TABLE	关于访问表的列的信息。
MULTIPROVIDER	联合所有具有相同分组和聚合的结果的操作符。

下面是查询计划中在 OPERATOR_NAME 显示的行引擎操作符列表。

ROW SEARCH	行引擎操作符的起始位置。OPERATOR_DETAILS 列出了投影的列。
LIMIT	限制输出行数的操作符。
ORDER BY	对输出行排序的操作符。
HAVING	过滤顶端谓词分组和聚集的操作符。
GROUP BY	分组和聚集的操作符。
MERGE AGGREGATION	合并多个并行的分组和聚合结果的操作符。
DISTINCT	消除重复的操作符。
FILTER	用谓词进行过滤的操作符。
UNION ALL	联合输入关系的操作符。
MATERIALIZED UNION ALL	生成中间结果物化的联合的操作符。
BTREE INDEX JOIN	通过B树的索引搜索联接输入关系的操作符。可以添加联接类型前缀。例如，B树索引左外联接显示为BTREE INDEX JOIN (LEFT OUTER)，无联接类型时代表内部联接。
CPBTREE INDEX JOIN	通过CPB树的索引搜索联接输入关系的操作符。可以添加联接类型前缀。
HASH JOIN	通过动态探测内置哈希表联接输入关系的操作符。可以添加联接类型前缀。
NESTED LOOP JOIN	通过嵌套循环联接输入关系的操作符。可以添加联接类型前缀。
MIXED INVERTED INDEX JOIN	无需使用倒排索引的格式转换联接行存储表和列存储表的操作符。可以添加联接类型前缀。
BTREE INDEX SEARCH	通过搜索B树的索引访问表。
CPBTREE INDEX SEARCH	通过搜索CPB树的索引访问表。
TABLE SCAN	通过扫描访问表。
AGGR TABLE	直接聚合基表的操作符。
MONITOR SEARCH	通过搜索监测视图的访问。
MONITOR SCAN	通过扫描监测视图的访问。

COLUMN SEARCH 是列操作符起始位置标识，而 ROW SEARCH 为行操作符的标识。
关于这一主题的更多信息，请参阅 *SAP HANA SQL Guide*。

3.5 使用 JDBC 驱动

为了从 JAVA 程序中访问 SAP HANA，你可以使用内置到程序中的 JDBC 驱动。其有如下属性：

- 100%纯JAVA
- 支持JDBC 4.0 标准扩展API(DataSource和ConnectionPoolDataSource)
- 类型4的驱动（使用SAP HANA的特定的网络协议建立数据库实例的连接）

3.5.1 安装驱动

可以使用命令分开安装驱动：

hdbinst.exe -a client (Microsoft Windows) or hdbinst -a client (Linux, UNIX).

安装完之后，驱动可以在 WINDOWS 平台的 C:\Program Files\sap\hdbclient\ngdbc.jar on Windows platforms 下找到，以及 Linux 和 UNIX 平台的 /usr/sap/hdbclient/ngdbc.jar。由于 SAP HANA Studio 是用 JAVA 编写，JDBC 驱动和 SAP HANA Studio 一起安装，可以通过安装获得。可以在如下地址找到：

Linux或UNIX平台：<SAP HANA Studio安装目录>/plugins/com.sap.ndb.studio.jdbc_<版本>.jar

Windows 平台：<SAP HANA Studio 安装目录>\plugins\com.sap.ndb.studio.jdbc_<版本>.jar

3.5.2 前提条件

你已经安装了JAVA平台：

- Sun Java JRE 1.4, or
- Sun Java JRE 1.6 or, for the application development Sun Java JDK 1.6 (for JDBC 4.0)

3.5.3 JDBC驱动整合

请为驱动文件ngdbc.jar的路径添加到CLASSPATH环境变量，以便Java平台可以找到并加载JDBC驱动程序。

3.5.4 加载JDBC驱动

请使用JAVA平台中Class类的forName方法。

`Class.forName("com.sap.db.jdbc.Driver");`

更多有关forName方法的信息，请访问sun.java.com上java平台的文档。

当执行Java应用程序时，Java平台自动在驱动管理器中加载并注册JDBC驱动程序。

3.5.5 连接地址

在连接地址中，你定义了连接数据库实例的属性。你可以在 JAVA 平台的文档中找到更多的信息，参见 sun.java.com。

语法:

jdbc:sap://<database_computer>[:<port>][/?<option1>[&<option2>]...] SAP HANA Database –

例子:

jdbc:sap://localhost:30115/?autocommit=false

使用这个连接地址，你在本地电脑上建立了数据库实例的连接。SAP 内存计算引擎使用的端口号为 30115，连接选项中把自动提交模式设置为 false。

3.5.6 JDBC 4.0 标准扩展 API

提供了访问服务器端数据源和 Java™ 编程语言处理的 API。JAVA API 的定义可以在 `javax.sql` 中找到。

这些类实现了 `javax.sql.DataSource` 和相应的 API。

3.5.7 JDBC 追踪

当你的应用程序通过 JDBC 连接到数据库时，你可以激活 JDBC 跟踪发现错误。

一个激活跟踪当前用户系统程序的简单方法是使用 SAP HANA studio。请右击 HDB 系统，选择 *Properties ► JDBC Trace*，然后单击 *Enable* 和 *ok*。

特点:

当 JDBC 追踪激活时，系统记录如下信息:

JDBC 程序调用的 JDBC API

JDBC API 调用的参数

执行的 SQL 语句和它们的结果。

前提:

你以启动（或即将启动）JDBC 程序的用户登录到操作系统。

注意:

你总是对所有的当前操作系统用户启动的 JDBC 程序激活 JDBC 跟踪。

配置对对所有的当前操作系统用户启动的 JDBC 程序有效。

过程:

变量 1:

1. 在命令行输入如下命令:

```
java -jar <installation_path>\ngdbc.jar
```

2. 选择 *Trace enabled*。
3. 选择你的追踪选项。

SAP MaxDB JDBC 追踪:

选项	命令行选项	描述
<i>Trace File Folder</i>	-	系统写入跟踪文件的目录。当你没有配置一个目录时，系统会把跟踪文件写到临时操作系统目录下（Microsoft Windows: %TEMP%, Unix和 Linux: \$TMP）

<i>Trace File Name</i>	TRACE FILENAME [<path>]<file_name>	设置跟踪文件名。系统分配给每个跟踪文件一个唯一的 ID.<id>: <file_name>_<id>.prt<file_name>默认值: jdbctrace
<i>Limit File Size</i>	TRACE SIZE <size> [KB MB GB]	限制跟踪信息的大小到<size>
-	TRACE SIZE UNLIMITED	移除所有的跟踪文件大小限制。
<i>Stop on Error</i>	TRACE STOP ON ERROR <error_code> TRACE STOP ON ERROR OFF	停止写入JDBC跟踪，当出现错误<error_code>或是发生第一个错误发生。

4. 选择 ok。

变量 2:

5. 显示当前设置，输入：

```
java -jar <installation_path>\sap\hdbclient\ngdbc.jar SHOW
```

6. 输入你的跟踪选项：

```
java -jar <installation_path>\sap\hdbclient\ngdbc.jar <command_line_option>
```

7. 输入

```
java -jar <installation_path>\sap\hdbclient\ngdbc.jar TRACE ON
```

变量 3:

在连接地址中指定下列选项：

trace=<file_name>

更多信息：Connection URL

结果：

系统把 JDBC 跟踪的结果写入到跟踪目录文件中。

3.5.8 匹配 SQL 和 JAVA 类型

通过 setObject 的转换

	TINYINT	SMALLINT	INTEGER	BIGINT	REAL	FLOAT	DOUBLE	DECIMAL	CHAR	VARCHAR	NCHAR	NVARCHAR	BINARY	VARBINARY	DATE	TIME	TIMESTAMP	BLOB	CLOB
String	✓	✓	✓																
java.math.BigDecimal	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							
Boolean	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							
Byte	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							
Short	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							
Integer	✓																		
Long	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							
Float	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓							
Double																			
byte[]													✓	✓					
java.sql.Date									✓	✓	✓	✓			✓		✓		
java.sql.Time									✓	✓	✓	✓				✓			
java.sql.Timestamp									✓	✓	✓	✓			✓	✓	✓		
Blob																		✓	
Clob																			✓

勾标记表示 JAVA 对象类型可以转换成 SQL 类型。该表显示了指定传给方法 `PreparedStatement.setObject` 或 `RowSet.setObject` 的目标 JDBC 类型参数的可能值。注意，某些转换会在运行时失败，如果该值是 `invalid.Labels` 的参数。

4 最佳实践

SAP HANA 的数据库的威力在于它前所未有的性能和易于管理。具有 SAP HANA 的数据库分析功能的用户将体验到很短的响应时间。响应时间的缩短，使用户能以高度互动的方式查询数据库，而不需要预定义的报告。因此，最佳实践力争最大限度地减少响应时间。

SAP HANA 数据库擅长高速计算。这在使用数据库的分析功能时，响应时间短是显而易见的，SAP HANA 通过快速提供结果的功能，以高度互动的方式查询数据库，而不需要预定义的报告或中间聚合表。

综合所说，SAP HANA 的性能很大程度上取决于如何使用它。在本节中，我们将重点放在最佳实践。最佳实践，对于性能而言，就是最小化数据库内的数据传输，并实现快速的响应时间。

4.1 列式存储引擎的特点

行和列存储引擎都有一个内置的优化各自的存储方法查询功能。因此每个存储引擎对于不同的任务有它的长处，行存储引擎支持由 SAP HANASQL 提供的所有功能，而列引擎优化了分析功能，支持更少的 SQL 特点，这些特点的速度远远快于行存储的相应的功能。由于列引擎并非支持所有由 SAP HANASQL 提供的功能，查询优化器有时会创建一个执行计划，该计划包括两个存储引擎，即使数据完全驻留在列存储。相反，它有时需要把行存储中间结果传输到列存储，因为随后的计算步骤该引擎可以更快的处理。作为一般的经验法则，我们建议您使用 SAP HANA Studio 的“Explain plan”，探索关键的 SQL 语句的执行和优化的数据模型和 SQL 语句。这个结果可以用来优化数据模型和相应的 SQL 语句。

下面的列表说明列引擎的查询功能。列引擎可以本地执行的查询语句，与下列格式表示的查询在逻辑上是等价的。

```
SELECT <列引擎由逗号分隔的表达式>
FROM <列引擎由逗号分隔的 FROM 定义>
[ WHERE <列引擎过滤条件> ]
[ GROUP BY <列引擎由逗号分隔的表达式> ]
[ HAVING <列引擎过滤条件> ]
[ ORDER BY <列引擎由逗号分隔的可选的 ASC | DESC 表达式> ]
[ LIMIT <任意表达式> [OFFSET <任意表达式> ] ]
```

列引擎的 FROM 定义可以是如下的一种：

- <表引用>
- (SELECT * FROM <表引用> WHERE <列引擎过滤条件>)
- ▣ 列引擎 FROM 定义 > INNER JOIN <列引擎 FROM 定义> ON <列引擎联接条件>
- <列引擎 FROM 定义> LEFT OUTER JOIN <列引擎 FROM 定义> ON <列引擎联接条件>
- <列引擎 FROM 定义> RIGHT OUTER JOIN <列引擎 FROM 定义> ON <列引擎联接条件>
- <列引擎 FROM 定义> FULL OUTER JOIN <列引擎 FROM 定义> ON <列引擎联接条件>

列引擎支持如下的联接选项：

- <一端的表引用> = <另一端的表引用>
 - 两列的数据类型应该是一样的，不同精度或规模的小数也被视为不同的数据类型。
- 列引擎联接条件之间的AND操作
 - 对于外联接，嵌套联接是不支持的。例如，如果T.a = S.a 和T.b = R.b已经在S和R之间有联接了，那么该语句是不支持的。
- 列引擎支持如下的过滤条件：
- 列引擎表达式之间的二进制比较(=, <>, <, >, <= and >=)
- 列引擎的LIKE, IS NULL and IN谓词
- 列引擎过滤条件AND, OR, NOT操作

列引擎支持如下的数量表达式：

- 常量值
- 列引擎表达式之间的二进制算术操作符(+, -, * and /)
- <列引擎表达式>
- 列引擎表达式之间的字符串级联(||)
- CASE <列引擎表达式> WHEN <列引擎表达式> THEN <列引擎表达式> ... [ELSE <列引擎表达式>] END
- CASE WHEN <列引擎过滤条件> THEN <列引擎表达式> ... [ELSE <列引擎表达式>] END
- 如下列出的函数作为列引擎表达式的参数
 - TO_DECIMAL, TO_NUMBER, TO_BIGINT, TO_REAL, TO_DOUBLE, TO_CHAR, TO_NCHAR, TO_DATE, TO_TIMESTAMP 和BINTOHEX/HEXTOBIN
 - 格式化字符串不支持TO_CHAR, TO_NCHAR, TO_DATE和TO_TIMESTAMP
 - LTRIM | RTRIM | TRIM, LENGTH, SUBSTR, INSTR和LOWER | UPPER
 - WEEKDAY, DAYS_BETWEEN, SECONDS_BETWEEN, ADD_DAYS, UTCTOLOCAL, LOCALTOUTC, ISOWEEK和QUARTER
 - LN | LOG, EXP | POWER | SQRT, SIN | COS | TAN, ASIN | ACOS | ATAN, SINH | COSH 和FLOOR | CEIL
 - NULLIF,和COALESCE
 - EXTRACT(YEAR | MONTH FROM <列引擎表达式>)
- 不带参数的函数由SQL解析器求值，计算结果以常量值传给列引擎。
 - CURRENT_CONNECTION, CURRENT_SCHEMA, CURRENT_USER, SYSUID, CURRENT_DATE/ CURRENT_TIME/ CURRENT_TIMESTAMP and CURRENT_UTCDATE/ CURRENT_UTCTIME/ CURRENT_UTCTIMESTAMP
- 与上面列出的表达式等价的表达式 (例如CAST函数)

列引擎支持如下的聚合表达式：

- MIN | MAX | COUNT | SUM | AVG ([DISTINCT] <列引擎表达式>)

由于SQL提供了多种可能的方式来制订查询，一条不在上面格式的SQL查询，如果该查询与格式正确的查询语句等价，就可以由列引擎在本地处理。下面是这种等价的例子。等价表示

为 \leftrightarrow 。

```
SELECT * FROM T, S WHERE T.a = S.a  $\leftrightarrow$  .SELECT * FROM T INNER JOIN S ON T.a = S.a
SELECT DISTINCT a, b, c FROM T  $\leftrightarrow$  .SELECT a, b, c FROM T GROUP BY a, b, c
SELECT * FROM T WHERE EXISTS (SELECT * FROM S WHERE T.a = S.a)  $\leftrightarrow$  .SELECT
DISTINCT T.* FROM T INNER JOIN S ON T.a = S.a
```

当表T只含有主键时等价

```
SELECT * FROM T WHERE NOT EXISTS (SELECT * FROM S WHERE T.a = S.a)  $\leftrightarrow$  .SELECT T.*
FROM T LEFT OUTER JOIN S ON T.a = S.a WHERE S.a IS NULL
```

4.2 SQL查询代价估计

SQL优化查询使用成本估算公式执行可能的替代方案。下表提供的系数列表，你可以用它来计算两个存储引擎支持的搜索操作相对执行时间。请注意，此表不能被用来计算实际的执行时间，因为它们是依赖于硬件参数，如CPU核心的数量或CPU和总线时钟速率。然而，该表可以用来比较替代方法，计算出最优的搜索策略或数据模型。

4.2.1 行式搜索代价模型

操作	模型	系数
投影	$C \times \langle \text{输出大小} \rangle$	$C = 0.821$
限制	$C \times \langle \text{输出大小} \rangle$	$C = 0.276$
分组/聚合	$C_1 \times \langle \text{输入大小} \rangle + C_2 \times \langle \text{输出大小} \rangle$	$C_1 = 2.063,$ $C_2 = 0.410$
选择	$C_1 \times \langle \text{输入大小} \rangle + C_2 \times \langle \text{输出大小} \rangle$	$C_1 = 0.521,$ $C_2 = 0.102$
嵌套循环联接	$C_1 \times \langle \text{左输入大小} \rangle \times \langle \text{右输入大小} \rangle + C_2 \times \langle \text{输出大小} \rangle$	$C_1 = 0.474,$ $C_2 = 0.410$
哈希联接	$C_1 \times \langle \text{左输入大小} \rangle + C_2 \times \langle \text{右输入大小} \rangle + C_3 \times \langle \text{输出大小} \rangle$	$C_1 = 0.728,$ $C_2 = 2.063,$ $C_3 = 0.675$
索引联接	$C_1 \times \langle \text{左输入大小} \rangle + C_2 \times \langle \text{左输入大小} \rangle \times \log(\langle \text{右输入大小} \rangle)$	$C_1 = 0.368,$ $C_2 = 0.028$
混合倒排索引联接	$C_1 \times \langle \text{左输入大小} \rangle + C_2 \times \langle \text{左输入大小} \rangle \times \log(\langle \text{右输入大小} \rangle) + C_3 \times \langle \text{输出大小} \rangle$	$C_1 = 0.423,$ $C_2 = 0.251,$ $C_3 = 0.190$
索引搜索	$C_1 \times \langle \text{输出大小} \rangle + C_2$	$C_1 = 0.496,$ $C_2 = 2.145$
整表扫描	$C \times \langle \text{输入大小} \rangle$	$C = 0.410$
远程访问	$C \times \langle \text{输出大小} \rangle$	$C = 8.986$

4.2.2 列式搜索代价模型

操作	模型	系数
联接	$\text{Weight} \times \langle \text{列引擎代价估计} \rangle$	$\text{Weight} = 0.0573368$

结果值物化	$C \times \langle \text{输出大小} \rangle$	$C = 0.481$
创建字典	$C \times \langle \text{输入大小} \rangle$	$C = 5.461$
分组/聚合	$C1 \times \langle \text{输入大小} \rangle + C2 \times \langle \text{输出大小} \rangle$	$C1 = 0.173,$ $C2 = 0.194$
分组/聚合(多列分组)	$C1 \times \langle \text{输入大小} \rangle + C2 \times \langle \text{输出大小} \rangle$	$C1 = 0.236,$ $C2 = 0.796$

4.3 SQL查询列引擎优化技巧

SAP HANA的列存储引擎对频繁使用的单块SPJG模式（选择、投影、联接和分组）进行了OLAP查询的优化。本节列出了执行SQL查询时，为了获得最佳的性能，列引擎最好避免的代价高的特性。

请注意，使用代价高的特性，不会对小表或从大表中提取的中间结果造成问题。然而对大型表或大型的中间结果集执行操作时，需要特别小心。这里建议的解决方案只是例子，不同程序之间的方案也是不一样的。

很多技巧建议避免使用列引擎不支持的本地操作。请参阅前一章查看本地支持的操作。在本章例子中所使用的表都为列存储表。

4.3.1 表达式

计算。如果使用的计算不被列引擎所支持，那么基于计算的操作都将被行引擎来执行，这会导致列引擎和行引擎之间大量中间结果集的传输。

例如，列引擎只支持TO_DATE函数无格式字符串。默认情况下，TO_DATE函数可以解析字符串为“YYYYMMDD”和“YYYY-MM-DD”。如果用户知道，源字符串可以由TO_DATE函数的解析，使用没有格式字符串的TO_DATE函数是更好的。否则，函数调用的参数都被转移到要处理的行引擎中。在下面的例子中，date_string是一个VARCHAR列。

速度慢的查询	SELECT * FROM T WHERE TO_DATE(date_string, 'YYYYMMDD') = CURRENT_DATE;
速度快的查询	SELECT * FROM T WHERE TO_DATE(date_string) = CURRENT_DATE;

使用列引擎本地支持的计算比使用非本地的计算快。然而，计算操作仍然比没有计算的操作慢，最好尽可能的避免进行计算。下面是一个如何避免计算的例子。

速度慢的查询	SELECT * FROM T WHERE (CASE WHEN a = 1 THEN 1 ELSE 2 END) = 2;
速度快的查询	SELECT * FROM T WHERE a <> 1 OR a IS NULL;

如果通过改变SELECT语句仍不可避免计算，在处理查询时还是可以避免的。这可以通过添加一个自动计算列实现，如下面的例子所示。添加一个自动生成的列提高了查询性能，同

时也增加插入和更新代价。

速度慢的查询

```
SELECT * FROM T WHERE b * c = 10;
```

速度快的查询

```
SELECT * FROM T WHERE bc = 10;
```

快速查询的DDL

```
ALTER TABLE T ADD (bc INTEGER  
GENERATED ALWAYS AS b * c);
```

请注意，数据插入后需要添加生成的列的列定义。这是必需的，以避免设置生成的列的值，如下面的示例所示。

添加生成列前插入

```
INSERT INTO T VALUES (1, 2,  
3);
```

添加生成列后插入

```
INSERT INTO T(a, b, c) VALUES  
(1, 2, 3);
```

隐式类型转换。SAP HANA可以进行隐式类型转换，即使用户没有明确指定一个类型转换操作。例如，如果有一个VARCHAR值和DATE值之间的比较，数据库系统执行隐式类型转换操作，把VARCHAR值转成一个DATE值。隐式类型转换是从低优先级类型转换到更高优先级的类型。你可以在SAP HANA SQL Reference找到类型优先级规则。

隐式类型转换是一种计算，并且和其他计算在性能上有相同的效果。出于这个原因，如果可能的话，最好避免进行隐式类型转换。如果两列经常被查询做比较，最好创建两列相同的数据类型。避免隐式类型转换代价的方法之一是使用显式类型转换。如果一个VARCHAR列和一个DATE列需要经常比较，程序员也知道把一个VARCHAR值转成DATE值可以产生期望的结果，为了节省代价，最好改变列值的类型而不是列的类型。例如，如果VARCHAR列包含日期值为“YYYYMMDD”的形式，它可以和从DATE值为“YYYYMMDD”生成的字符串进行比较。

速度慢的查询

```
SELECT * FROM T WHERE  
date_string <  
CURRENT_DATE;
```

速度快的查询

```
SELECT * FROM T WHERE  
date_string <  
TO_CHAR(CURRENT_DATE  
, 'YYYYMMDD');
```

如果不能直接避免隐式类型转换，以增加插入和更新的代价添加生成的列可能会提供一个更有效的解决方法。例如，用户可以使用下列查询，找到'1', '1.0', '1.00'存储在一个VARCHAR列。

在下面的例子中，s是一个VARCHAR列。

速度慢的查询

```
SELECT * FROM T WHERE s = 1;
```

速度快的查询

```
SELECT * FROM T WHERE n = 1;
```

快速查询的DDL

```
ALTER TABLE T ADD (n DECIMAL  
GENERATED ALWAYS AS s);
```

4.3.2 联接

非等值连接谓词。列引擎本身并不支持除了等于条件以外的其他联接谓词。换句话说，列引擎本身只支持等值连接。连接谓词连接或笛卡尔积、比较、没有联接谓词的联接不被支持。在这些情况下，行引擎会被激活，并且中间数据会传送到该引擎。如果只用于非等值连接谓词，行引擎将使用嵌套循环算法执行联接操作。使用嵌套循环算法以及格式化中间数据，如

果中间结果集很大，代价会很昂贵。对于外连接，如果等值联接谓词由AND与非等值连接谓词连接，他们将以非等值连接谓词相同的方式处理。对于内连接，如果等值联接谓词与非等值连接谓词连接，只有等值联接谓词是用来联接，而非等值连接谓词只是用于过滤。在这种情况下，即使作为一个整体的联接谓词选择性不够，如果同等联接谓词本身是选择性不够，联接操作可能需要相当长的时间。

如下所示是把非等值连接谓词改写成等值连接谓词的一个例子。在这个例子中，M是一个，包含利息月的第一个和最后一天的表。

速度慢的查询

```
SELECT M.year, M.month,  
SUM(T.ship_amount) FROM T JOIN M ON  
T.ship_date BETWEEN M.first_date  
AND M.last_date GROUP BY M.year,  
M.month;
```

速度快的查询

```
SELECT M.year, M.month,  
SUM(T.ship_amount) FROM T JOIN M ON  
EXTRACT(YEAR FROM T.ship_date) =  
M.year AND EXTRACT(MONTH FROM  
T.ship_date) = M.month GROUP BY  
M.year, M.month;
```

计算列的谓词。列引擎本身不支持计算列的同等联接谓词。如果列引擎本身不支持计算，包含计算的子操作的中间结果将会被格式化并再次被列引擎使用。如果列引擎本身不支持计算，从两个子操作的中间结果被格式化，并被行引擎使用。在任何情况下，如果中间数据量很大，将会对性能有影响。一个避免这样的计算方式的办法是使用生成的列。下面是一个避免计算使用生成的列联接列的例子。

速度慢的查询

```
SELECT M.year, M.month,  
SUM(T.ship_amount) FROM T JOIN M ON  
EXTRACT(YEAR FROM T.ship_date) =  
M.year AND EXTRACT(MONTH FROM  
T.ship_date) = M.month GROUP BY  
M.year, M.month;
```

速度快的查询

```
SELECT M.year, M.month,  
SUM(T.ship_amount) FROM T JOIN M ON  
T.year = M.year AND T.month =  
M.month GROUP BY M.year, M.month;
```

快速查询的DDL

```
ALTER TABLE T ADD (year INTEGER  
GENERATED ALWAYS AS EXTRACT(YEAR  
FROM T.ship_date)); ALTER TABLE T  
ADD (month INTEGER GENERATED  
ALWAYS AS EXTRACT(MONTH FROM  
T.ship_date));
```

外联接的过滤谓词。列引擎本身不支持内外部联接谓词的过滤谓词。对于右侧左外连接的过滤，和左侧右外联接的过滤谓词是例外，因为这些谓词联接操作的调用产生相同的结果。

循环联接。列引擎本身并不支持联接树，如果一个外联接包含在循环内。如果有一个涉及外联接的循环，完成循环的子联接的结果会被物化，以打破循环。列引擎本身支持循环内联接，但最好避免使用他们，因为其表现逊于非循环内部联接。打破这种循环的方法之一是通过改变架构，把联接涉及的列移动到不同的表中。下面是一个例子。对于例子中的非循环联接，SUPPLIER 表的 NATION 列移动到了表 LINE_ITEM 中。

循环联接

```
SELECT * FROM supplier S,  
customer C, line_item L WHERE  
L.supp_key = S.key AND  
L.cust_key = C.key AND S.nation  
= C.nation;
```

非循环联接

```
SELECT * FROM supplier S,  
customer C, line_item L WHERE  
L.supp_key = S.key AND  
L.cust_key = C.key AND  
L.supp_nation = C.nation;
```

过滤访问多个表外连接的谓词。列引擎本身不支持多个表的过滤谓词，如果他们出现在外联接中。如果存在这种过滤谓词，包含谓词的子结果会在执行前被物化。对于左外连接的左孩子的过滤谓词和对右外连接的右孩子的过滤谓词是例外，因为向上移动这些谓词到外部联接会产生相同的结果，这样的移动将由 SQL 优化器自动执行。

下面是一个过滤谓词触发物化中间结果的例子。一种避免例子中物化的方法是维护表 LINE_ITEM 表的 PRIORITY 列，而非 ORDERS 表。

```
SELECT * FROM customer C LEFT JOIN (SELECT * FROM orders O JOIN lineitem  
L ON O.order_key = L.order_key WHERE L.shipmode = 'AIR' OR O.priority  
= 'URGENT' ) ON C.cust_key = L.cust_key
```

外联接的常数和计算值投影。列引擎本身不支持在外联接下的常数或计算值投影。如果存在这样的常数或计算值投影，包含投影的子结果会在执行前被物化。对于左外连接的左孩子的过滤谓词和对右外连接的右孩子的常数和计算值是例外，因为把投影向上移动到联接会产生相同的结果，这样的移动将由 SQL 优化器自动执行。一个避免物化中间结果的解决办法，是添加一个常数或计算值生成的列。

当联接涉及多列时，列引擎试图动态创建一列连接列来处理联接。连接列是一个新列，内部包含涉及列的所有信息。如果查询语句没有在更新事务中使用或有一个与其他更新事务创建连接列上的锁冲突，连接列将不被创建。当没有创建连接列时，必须使用低效率的执行策略。

为了避免运行时连接列创建的开销或由于连接列创建失败所执行的次优策略，建议事先建立所需的列。下面是一个需要连接列和 DDL 以创建所需的连接列查询的例子。

需要连接列的查询

```
SELECT M.year, M.month,  
SUM(T.ship_amount) FROM T JOIN M  
ON T.year = M.year AND T.month =  
M.month GROUP BY M.year, M.month;
```

创建所需连接列的DDL语句

```
CREATE INDEX T_year_month ON
```

```
T(year, month); CREATE INDEX  
M_year_month ON M(year, month);
```

4.3.3 EXISTS / IN 谓词

不相交 EXISTS 谓词。当一个 EXISTS 或 NOT EXISTS 谓词通过 OR 与其他谓词相连，其会内部映射到左外连接。由于处理左外连接的代价比内联接大，建议尽可能避免使用不相交的 EXISTS 谓词。下面是一个如何避免 EXISTS 谓词的例子。

速度慢的查询语句

```
SELECT * FROM T WHERE EXISTS (SELECT  
* FROM S WHERE S.a = T.a AND S.b =  
1) OR EXISTS (SELECT * FROM S WHERE  
S.a = T.a AND S.b = 2);
```

速度快的查询语句

```
SELECT * FROM T WHERE EXISTS (SELECT  
* FROM S WHERE S.a = T.a AND (S.b =  
1 OR S.b = 2));
```

访问多表的 NOT EXISTS 中的过滤谓词。由于 NOT EXISTS 是由左外联接处理，适用于外联接的调优技巧同样也适用于 NOT EXISTS。

访问多表的不相交 EXISTS 中的过滤谓词。由于不相交的 EXISTS 是由左外联接处理，适用于外联接的调优技巧同样也适用于不相交的 EXISTS。

NOT IN 谓词。由于处理 NOT IN 比 NOT EXISTS 代价大，推荐尽量使用 NOT EXISTS 而不是 NOT IN。下面是一个避免使用 NOT IN 谓词的例子。请注意，例子中的转换一般情况下不是有效的，只有在感兴趣的列中没有空值时才是有效的。如果 NOT NULL 没有显示声明，SQL 优化器对所有的列自动执行转换。

NOT IN 查询

```
SELECT * FROM T WHERE a NOT IN (SELECT  
a FROM S);
```

某些情况下等价的查询

```
SELECT * FROM T WHERE NOT EXISTS  
(SELECT * FROM S WHERE S.a = T.a);
```

4.3.4 集操作

UNION ALL / UNION / INTERSECT / EXCEPT。由于列引擎本身不支持 UNION ALL、UNION、INTERSECT 和 EXCEPT，避免使用它们可以提高性能。下面是如何避免 UNION，INTERSECT 和 EXCEPT 的例子。请注意，例子中的转换一般情况下不是有效的，只有在感兴趣的列中没有空值时才是有效的。

UNION 查询

```
SELECT a, b FROM T UNION SELECT  
a, b FROM S;
```

某些情况下等价的查询

```
SELECT DISTINCT COALESCE(T.a,  
S.a) a, COALESCE(T.b, S.b) b FROM  
T FULL OUTER JOIN S ON T.a = S.a  
AND T.b = S.b;
```

INTERSECT 查询

```
SELECT a, b FROM T INTERSECT  
SELECT a, b FROM S;
```

某些情况下等价的查询

```
SELECT DISTINCT T.a a, T.b b FROM
```

EXCEPT 查询

某些情况下等价的查询

```
T JOIN S ON T.a = S.a AND T.b = S.b;  
  
SELECT a, b FROM T EXCEPT SELECT  
a, b FROM S;  
  
SELECT DISTINCT T.a a, T.b b FROM  
T WHERE NOT EXISTS (SELECT * FROM  
S WHERE T.a = S.a AND T.b = S.b);
```

4.4 SQLScript 推荐实践

以下的优化技巧适用于所有的 SQLScript 语句。这里提出的优化覆盖了数据流如何利用 SAP HANA 数据库的并行。

4.4.1 降低 SQL 语句的复杂度

SQLScript 变量让你能把一个复杂的 SQL 语句随意分解成许多简单的语句，这使得 SQLScript 存储过程更容易理解。为了说明这一点，考虑下面的查询：

```
books_per_publisher = SELECT publisher, COUNT (*) AS cnt FROM :books GROUP BY publisher;  
largest_publishers = SELECT * FROM :books_per_publisher WHERE cnt >= (SELECT MAX (cnt)  
FROM :books_per_publisher);
```

把这条查询写成简单的 SQL 语句需要定义一个临时视图(使用 WITH)或是多次重复子查询。上述两条语句把复杂查询分成两个通过表变量连接的简单 SQL 语句。这条查询更易理解，因为表变量的名字表达了查询的意思，并且他们把复杂的查询分成更小的逻辑块。

SQLScript 编译器将这些语句组合成一个单一的查询，或使用表变量提示常见的子表达式。生成的应用程序在不牺牲性能的情况下更容易理解。

4.4.2 识别共同的子表达式

在之前的章节中包含了查询中共同的子表达式。这种常见的子表达式可能引进代价很大的重复计算，应尽量避免。对于查询优化器而言，检测 SQL 查询中共同子表达式是很复杂的。如果把一个复杂的逻辑分解成逻辑子查询，它可以帮助优化器确定共同的子表达式，以获得更高效的执行计划。如果有疑问，你可以利用 EXPLAIN 研究 HDB 如何对待特定的语句。

4.4.3 多层聚合

计算多层聚合可以通过使用分组集实现。这种方法的优点是多层次的分组可以在单条 SQL 语句中计算。

```
SELECT publisher, name, year, SUM(price) FROM :it_publishers, :it_books WHERE  
publisher=pub_id AND crcy=:currency GROUP BY GROUPING SETS ((publisher, name, year),  
(year))
```

要获得不同的多层次分组，客户端通常需要反复检查结果，例如对分组属性过滤 NULL。在特殊情况下的多层次聚合，SQLScript 可以利用粗聚合计算的细分组结果以及返回表变量中分组的粒度。

```
books_ppy = SELECT publisher, name, year, SUM(price) FROM :it_publishers, :it_books WHERE
```

```
publisher = pub_id AND crcy = :currency GROUP BY publisher, name, year;  
books_py = SELECT year, SUM(price) FROM :books_ppy GROUP BY year;  
这与开发人员的选择问题有关，以最适合应用的需求为宗旨。
```

4.4.4 理解语句执行的代价

重要的是要记住，即使 SAP HANA 的数据库是一个内存数据库引擎、操作速度快，每个操作都有其相关的开销和一些比其他数据库代价更大的操作。

举个例子，计算两个结果集的 UNION ALL 的开销比计算相同结果集的 UNION 小，因为 UNION 会执行消除重复记录的操作。计算引擎的 CE_UNION_ALL 操作符（也包括 UNION ALL）把两个输入表堆到对方，通过使用引用，无需在内存中移动任何数据。相反，作为 UNION 的一部分，消除重复记录需要对数据排序或是哈希来实现消除重复，以及数据的物化。存在有很多类似的例子。因此，重要的是要意识到这些问题，如果可能的话，避免这些昂贵的操作。一般 HDB 提供语句来检查 SQL 语句的开销（有关详细信息，请参阅 SQL Reference Documentation）。

你可以从视图 SYS. QUERY_PLANS 中获得查询计划，该视图由所有用户共享。以下是从视图读取查询计划的一个例子。

```
EXPLAIN PLAN [ SET PLAN_ID = <plan_id> ] FOR <dml_stmt>  
SELECT lpad(' ', level) || operator_name AS operator_name,  
operator_details, object_name, subtree_cost, input_cardinality,  
output_cardinality, operator_id, parent_operator_id, level, position  
FROM sys.query_plans WHERE PLAN_ID = <plan_id> ORDER BY operator_id;
```

有时相同的查询替代的公式可能会导致更快的响应时间。因此重新组建关键性能查询语句并且检查计划会导致更好的性能。

HDB 提供了一个应用程序级的功能库，处理频繁的任务，例如货币转换。这些功能执行起来开销很大，因此在调用函数之前，尽可能减少输入是很有意义的。

4.4.5 利用底层引擎

SQLScript 可以利用内置的函数或 SQL 语句的特定功能。举例来说，如果我们的数据模型是一个星型模式，把数据建模成分析视图是很有意义的。这使 HDB 在计算联接时能利用星型模式，来产生更好的性能。

同样，如果应用程序涉及复杂的联接，把数据建模成属性视图是很有意义的。再次，这传达了 HDB 计算联接时使用的数据结构更多的信息。下面你可以找到一个如何在各种实现逻辑方式之间选择的建议。图中 SQLScript 只出现在底部的叶子节点。使用 CE 功能或者只在存储过程中的 SQL 语句，仍然允许可以在底层数据库系统的许多优化。但是使用命令式结构的 SQLScript 程序被其他程序调用时，例如早期谓词过滤数据，就不再适用。当性能至关重要时，必须仔细分析使用这些结构所产生的影响。最后请注意，如果没有把 SQL 查询的结果分配到一个表变量，那么结果会直接作为结果集返回到客户端。这可能是非常有效的，因为该结果在返回给客户端之前，不需要在服务器上物化。

4.4.6 减少依赖

HDB 加快处理的最重要方法之一是大规模并行执行查询。尤其，并行利用了多层次的粒度：例如，不同用户的要求可以并行处理，并且查询语句中的单个关系操作符并行地在多个处理器上执行。它也可以并行执行的单一 SQLScript 不同的报表，如果这些陈述是相互独立的。

如果 SQLScript 语句间是相互独立的，并行执行单一语句也是可能的。记住 SQLScript 被翻译成数据流图，并可以在该图的独立路径上并行执行。

从一个 SQLScript 开发人员的角度来看，我们可以尝试并行执行，避免单独的 SQL 语句之间的不必要的依赖，如果可能的话，也可以在支持的数据库引擎使用声明的结构。前者是指避免变量的引用，而后者意味着避免命令式的功能，如游标等。

4.4.7 模拟 SQL 语句中的函数调用

CALL 语句不能在 SQL 中使用，所以，每当我们调用输入表转换结果的存储过程，中间结果必须绑定到一个表变量中。

在特殊情况下的一个只读的返回一个单一的表，你可以用 WITH RESULT VIEW 创建一个存储过程，然后就可以调用嵌入在如下 SQLSELECT 语句的过程（有关详细信息，见此文件的上述讨论）：

```
CREATE PROCEDURE ProcWithResultView(IN id INT, OUT o1 CUSTOMER) LANGUAGE SQLSCRIPT
READS SQL DATA WITH RESULT VIEW ProcView AS
BEGIN o1 = SELECT * FROM CUSTOMER WHERE CUST_ID = :id;
END;
```

然后，就可以把存储过程的结果作为 SQL 语句的一部分：

```
SELECT * FROM ProcView WITH PARAMETERS ('placeholder' = ('$id$', 5))
```

4.4.8 避免混合使用计算引擎操作符和 SQL 查询

SQL 查询和计算引擎操作使用的关系操作符的语义是不同的。在计算引擎中，操作将被数据流图上执行的查询实例化。因此查询可以显著地改变数据流图的语义。

例如，考虑使用出版商属性（而不是年）查询的计算视图，包含一个聚合节点（如 CE_AGGREGATION）定义在出版商和年份上。对年份的分组将会从分组中移除。显然，这降低了分组的粒度，从而改变了该模型的语义。另一方面，包含出版商和年份的分组的嵌套 SQL 语句中，如果封闭的查询语句只查询出版商，聚合层次将不会改变。

目前限于所列的不同语义，将会限制使用混合的两种类型的数据流的优化。因此，应避免在一个程序中混合使用两种类型的操作。

4.4.9 避免使用游标

虽然有时需要使用游标，它们意味着一次一行的处理，因此，会错过 SQL 引擎优化的机会。所以，你应该考虑更换使用如下 SQL 语句的循环游标：

只读访问：

对于只读访问，考虑使用简单的选择或者联接：

```
CREATE PROCEDURE foreach_proc LANGUAGE SQLSCRIPT AS val
decimal(34,10) := 0; CURSOR c_cursor1 FOR SELECT isbn, title, price
FROM books;
BEGIN
FOR r1 AS c_cursor1 DO val := :val + r1.price; END FOR;
END;
```

总和也可以由 SQL 引擎来计算:

SELECT sum(price) into val FROM books;

在 SQL 引擎中计算聚合将使得 SQL 在多个处理器上并行执行。

更新和删除:

对于更新和删除, 考虑使用 **WHERE** 语句:

```
CREATE PROCEDURE foreach_proc LANGUAGE SQLSCRIPT AS val INT := 0;
CURSOR c_cursor1 FOR SELECT isbn, title, price FROM books;
BEGIN
FOR r1 AS c_cursor1 DO IF r1.price > 50 THEN DELETE FROM Books WHERE
isbn = r1.isbn; END IF; END FOR;
END;
```

该删除语句也可以由 SQL 引擎计算:

DELETE FROM Books WHERE isbn IN (SELECT isbn FROM books WHERE price > 50);

在 SQL 引擎中计算该值减少了通过 HDB 运行时堆栈的调用, 并且潜在地从内部优化机制如缓存或是并行执行中获益。

插入表

```
CREATE PROCEDURE foreach_proc LANGUAGE SQLSCRIPT AS val INT := 0;
CURSOR c_cursor1 FOR SELECT isbn, title, price FROM books;
BEGIN
FOR r1 AS c_cursor1 DO IF r1.price > 50 THEN INSERT INTO ExpensiveBooks
VALUES(..., r1.title, ...); END IF; END FOR;
END;
```

该插入语句也可以由 SQL 引擎计算:

SELECT ..., title, ... FROM Books WHERE price > 50 INTO ExpensiveBooks;

与更新删除操作类似的是, 在 SQL 引擎中计算该值减少了通过 HDB 运行时堆栈的调用, 并且潜在地从内部优化机制如缓存或是并行执行中获益。

4.4.10 避免使用动态 SQL

动态 SQL 是一个表达应用程序逻辑非常强大的方式, 其允许在存储过程运行的时间内构建 SQL 语句。然而, 执行动态 SQL 是缓慢的, 因为编译时必须对每次调用检查和查询优化。所以当有使用变量的动态 SQL 的替代方案时, 应当用来使用。

另一个相关的问题是安全, 因为没有对使用的变量进行适当检查, 构建 SQL 语句可能会创

造一个安全漏洞，如通过 SQL 注入。在 SQL 语句中使用变量避免了这些问题，因为在编译时类型检查和参数不能注入任意 SQL 代码。

总结的动态 SQL 可能使用场景如下：

功能	建议方案
Projected attributes	Dynamic SQL
Projected literals	SQL + variables
FROM clause	SQL + variables; result structure must remain unchanged
WHERE clause values	SQL + variables
WHERE clause – attribute names & Boolean operators	Dynamic SQL

4.4.11 跟踪和调试

当对创建 SQLScript 程序审查问题时，可能要增加 indexserver 的跟踪级别。在这种情况下，HDB 将不仅追查错误消息，也会产生更详细的信息。

对于 SQLScript 以下的跟踪信息（如诊断文件）是有关的：

- Component Indexserver
calcengine, calcengineinstantiate, llang, or SQLScript
这把跟踪的信息写到文件indexserver_<host>.<port>.<xyz>.trc
- Component SQL

这把SQL命令的跟踪信息写到文件SQLtrace_<host>_<port>_<xyz>.py

- Performance Trace

此外，indexserver.ini 配置可以用来获得 SQLScript 运行时实例化的计算引擎模型的更详细信息。

- calcengine部分
 - 记录: tracemodels = yes
如果设成yes，优化前和优化后的json模型将会打印到跟踪文件
 - 记录: show_intermediate_results = yes or topXXX
TopXXX会写入结果集的前面xxx行，yes写入所有结果
- trace部分
 - Entry: row_engine = warning
该记录设置HDB引擎的跟踪级别，启用SQLSCRIPT编译器的警告功能，包括过时的警告信息，默认级别为错误。
- 运行时数据显示在HDB工作室管理视图的performance标签