面向工业时序数据管理与分析融合的数据模型转换

李烁麟[1]，林欣涛[1]，田原[2]，许京奕[3]，陈荣钊[4]，李永瑾[3]，乔嘉林[4]
1. 清华大学软件学院，北京 100084；
2. 天谋科技（上海）有限公司，上海 201824；
3. 北京大数据先进技术研究院，北京 100195；
4. 天谋科技（北京）有限公司，北京 100192

摘要
工业物联网中，设备监控数据通常以时序数据形式存储，并采用多层级路径的形式进行建模，便于点位管理与访问，但该方式难以直接支持关系模型的多维分析，往往需要复杂的ETL转换步骤。针对工业时序数据的管理与分析需求在模型层面的矛盾，提出一种融合的时序数据模型，采用层次化的树模型进行灵活数据写入和存储，并通过表视图提供多维数据分析能力。本方法在工业物联网时序数据库Apache IoTDB中进行了系统性实现，使得一份数据能够同时服务于工业监控与分析场景。

关键词
工业物联网；时序数据库；数据模型；元数据管理；层次数据；模型转换

中图分类号：TP311 文献标志码：A doi：10.11959/j.issn.2096-0271.20260XX

A data model transformation approach for integrated management and analysis of industrial time-series data

LI Shuolin[1], LIN Xintao[1], TIAN Yuan[2], XU Jingyi[3], CHEN Rongzhao[4], LI Yongjin[3], QIAO Jialin[4]
1. School of Software, Tsinghua University, Beijing 100084, China
2. Timecho Ltd. (Shanghai), Shanghai 201824, China
3. Advanced Institute of Big Data, Beijing, Beijing 100195, China
4. Timecho Ltd. (Beijing), Beijing 100192, China

Abstract
In industrial Internet of Things (IIoT) scenarios, device monitoring data are typically stored in the form of time-series data and modeled using hierarchical path structures to facilitate point management and access; however, this modeling approach does not directly support multidimensional analysis based on relational models and often requires complex ETL transformations. To address the model-level conflict between time-series data management and analysis in industrial settings, this paper proposes a unified time-series data model. The model leverages a hierarchical tree structure for flexible data ingestion and storage, while offering multidimensional analytical capabilities through a classical relational model view. We implement this approach systematically in the industrial time-series database

Apache IoTDB, enabling a single dataset to support both monitoring and analytical requirements.

Key words
industrial internet of things, timeseries database, data model, schema management, hierarchical data, data model transformation

0 引言

在工业物联网（industrial internet of things，IIoT）[1]快速发展的当下，时序数据[2]作为设备物理量的数字化记录，在设备状态监控、故障诊断与预警、工艺参数优化等关键领域发挥着至关重要的作用。这些数据承载着工业生产过程中的实时动态信息，是工业数字化、智能化的核心基石[3]。

在工业众多应用场景中，监控场景是时序数据的主要应用场景之一，且应用最广泛。在此场景下，时序数据通常采用多层级命名的监测点位进行建模和管理，如电力系统的KKS[4]编码、建模传感器网络的监视控制与数据采集（supervisory control and data acquisition，SCADA）[5]系统等。这种层次化的树模型能够直观地映射工业现场的设备结构和测点分布，极大地满足了点位灵活新增和实时读取的需求，为工业现场的实时监控提供了高效、便捷的支持，成为工业监控系统的重要数据模型选择。

然而，随着工业智能化进程[6]的不断推进，企业对数据分析的需求日益增强。传统的树模型在面对复杂的数据分析任务时，逐渐显露出其局限性。数据分析需要基于关系模型[7]的多维分析能力，这种能力能够支持更灵活、更深入的数据挖掘和业务洞察。但在现有方案中，为了同时满足监控和分析的需求，往往需要部署两类不同的时序数据库[8]：一类是具备层次化树模型的时序数据库，用于满足监控场景的实时性和点位管理需求；另一类是具备关系模型的数据仓库，以支持数据分析场景的多维分析要求。这两种数据库之间需要通过复杂的ETL[9]过程进行数据模型的转换，这不仅带来了多份存储成本，增加了系统的硬件和存储资源消耗，还显著提升了系统的复杂度，导致数据一致性难以保证，数据处理流程变得冗长且容易出错，给业务的高效开展带来了诸多挑战。

针对工业时序数据在管理与分析需求层面存在的模型矛盾，如何在同一套数据体系中同时满足监控和分析的不同需求，成为工业物联网领域亟待解决的关键问题。为此，本文提出一种融合的时序数据模型，旨在打破传统方案中数据采集管理和分析的割裂状态。该模型采用层次化的树形模型进行灵活的数据写入和存储，充分保留树模型在监控场景下的优势，确保工业现场数据的实时采集和点位管理的高效性；同时，通过表视图提供多维数据分析能力，满足数据分析场景对复杂查询和深度分析的需求。这种融合模型使得一份数据能够同时服务于工业监控与分析场景，避免了数据的重复存储和复杂的模型转换过程。

本文的研究工作在工业物联网时序数据库Apache IoTDB[10]（后文用IoTDB指代）中进行了系统性实现，通过对IoTDB进行改进和优化，使其同时具备了支持树模型读写和表视图分析的能力，实现了从数据采集、存储到分析的全流程贯通。该方案在多个工业企业进行了实际落地验证，

有效解决了工业企业在数据管理和分析过程中面临的成本高、复杂度高等问题，为工业物联网的智能化发展提供了新的思路和技术支撑。

本文的主要贡献有以下3个方面。

• 在树模型上构建支持多维分析的表视图机制：引入表视图的逻辑抽象，允许用户能够基于业务需求构建具有关系语义的时序表视图，支持使用SQL实现复杂的分析操作。

• 提出一种基于路径语义的树模型向表视图的转换方法：设计了从层次结构的树模型中提取设备标识信息和设备测点集合并映射为二维表结构的算法，实现了树模型结构向表视图的转换，为统一支持监控与分析提供了基础。

• 在IoTDB中实现该机制并验证其实用性：在IoTDB系统中实现了表视图创建与视图查询，并结合标准基准测试工具TPCx-IoT进行评测。测试内容覆盖数据写入、点查询、范围查询、聚合查询以及复杂查询等场景。实验结果表明，引入表视图对树模型的写入性能影响极小，吞吐量下降不足1%。在范围、聚合与复杂查询中，表视图与树模型的性能接近，表视图相比树模型耗时不超过15%，这表明在获得关系型查询能力的同时，额外开销处于可接受范围，在实际工业场景中具备可行性。

1 相关工作

数据模型[11]是描述数据库中数据存储、组织与操作的概念，是数据库设计的核心基础。使用良好的数据模型进行数据建模，设计合理的数据组织和操作方式，决定了数据管理的语义能力和执行能力，是保证高效数据管理的基础。

在时序数据库中，不同的数据模型直接影响系统在数据采集、存储、查询分析等环节的能力与性能。对于以写入为主、模式演化频繁的时序场景，时序数据库通常选择灵活的非关系型模型以简化数据接入与扩展。然而在分析与查询阶段，越来越多的系统呈现出向关系型计算范式融合的趋势，其根本原因在于关系型模型在多维分析、聚合运算和关联查询方面具备成熟的表达能力与工具生态。

1.1 树模型时序数据库

在物联网时序场景中，数据通常先于模式定义产生，这与传统关系型数据库依赖完整的预定义模式显著不同。为了适应这种特性，树状数据模型以较低的抽象层次组织数据，使其能够自然而直观地映射现实世界的层次化结构，如“位置-设备-测点”。

IoTDB是使用树模型[12]的时序数据库代表，它将时间序列以层次结构组织为一颗树，与实际工业生产设备的层次组织结构相互映射。

图1展示了IoTDB树模型对实际场景的建模示例。IoTDB树模型中的节点可以

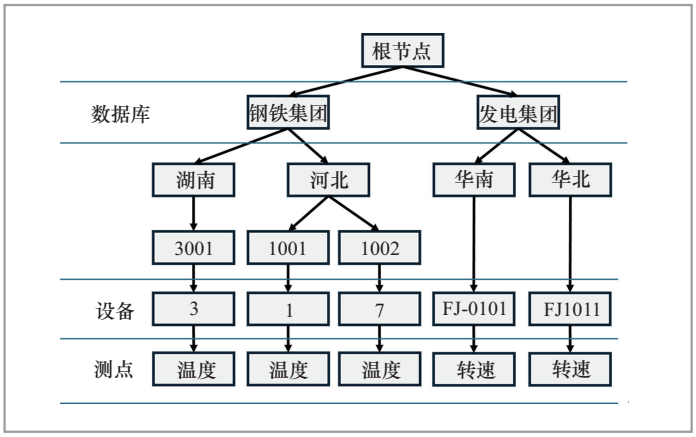


图1 IoTDB树模型建模示例

自根节点向下划分为3种类型

• 数据库节点：系统会为其子树分配存储与计算资源。

• 设备节点：每个设备节点对应实际场景中的一个物理设备。

• 测点节点：每个测点节点对应实际场景中的一个传感器，测点节点不仅需要记录传感器实时产生的时序数据，还要记录数据类型、编码压缩方式等元数据信息。

以测点为基本管理单位的模式定义使树模型能支持灵活模式扩展，在边缘设备的数据采集和动态模式演化任务中表现出较好的适应性，但这也使树模型的元数据语义管理能力受限：尽管使用路径标识可对设备进行唯一定位，但是路径中每个层级的语义类型（包括地域、车间、设备类型等）不能被结构化地表达，缺乏显式的字段标识。正是缺乏结构化的定义，导致树模型难以支持数据分析场景中针对设备的复杂分析、维度建模和跨序列整合等方面的分析任务。

1.2 关系型时序数据库

关系模型具备较好的结构化表达能力和强大的查询分析能力，广泛用于企业信息化系统。然而其建模抽象层次较高，对结构不规整、演化频繁[13-14]的时序数据管理存在一定局限。一方面，关系模型通常依赖预定义的模式，难以适应动态设备接入和测点变更；另一方面，在设备多样、元数据异构的工业物联网环境中，频繁的模式演化会给系统维护带来较大挑战。

关系型时序数据库库通常遵循“关系表+时序表”的元数据建模思路，即先在关系表当中存储时间序列的标识信息（例如设备编号、所属区域等），然后将各个设备的时序数据写入时序表中。

以**图2**所示的TimeScaleDB[15]建模方法为例，它在PostgreSQL[16]的基础上引

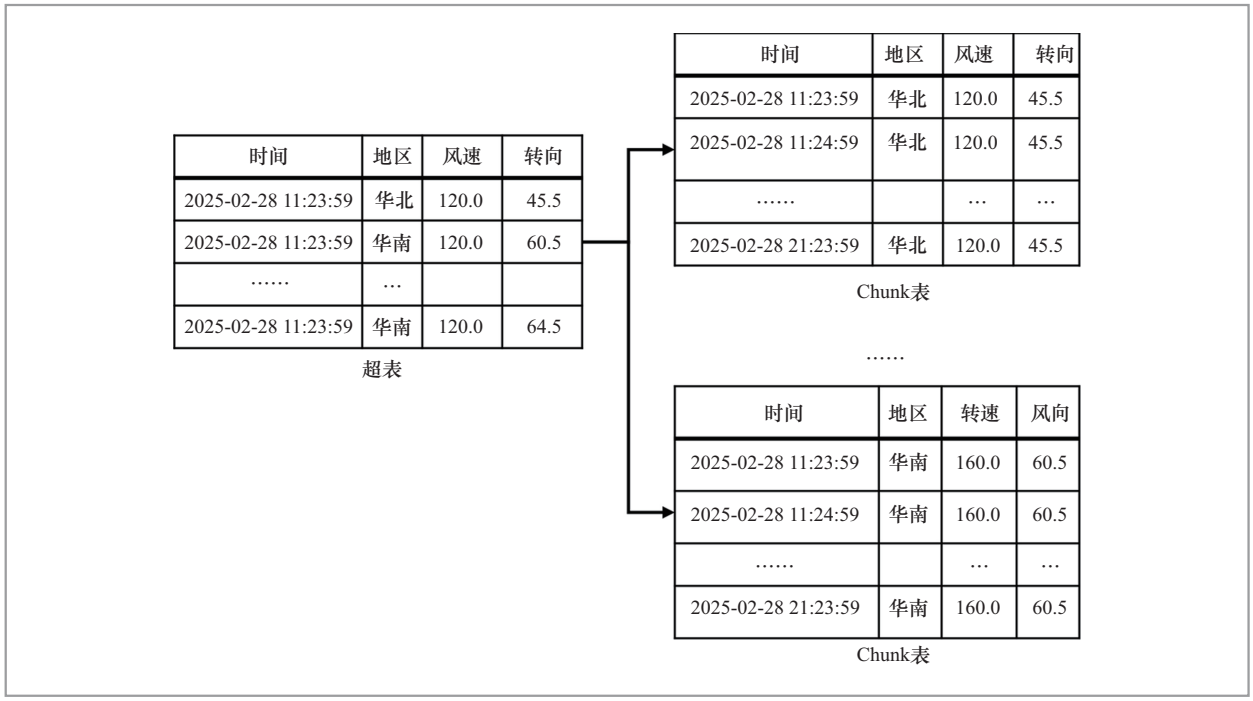


图2 超表建模示例

入“超表”的概念，用户以其为入口进行标准SQL的时序数据操作。系统会根据时间列（及可选空间标签列）将数据自动划分为多个物理子表（chunk），每个物理子表都是一个独立的 PostgreSQL 表，并为其建立局部索引。

需要指出的是，超表具有视图的性质，其本质上是对存储为关系型模型的时序数据进行逻辑聚合，并提供统一访问的关系型抽象，这与本文工作（即对树模型时序数据进行表视图映射）存在关键区别。超级表具备关系模型的优势（结构化表达能力），但同时受限于关系模型在时序与物联网场景中的天然不足，例如模式演化成本较高、对动态设备接入和多样化元数据的适配性有限，无法支持工业层次化树形建模的场景。

1.3 基于标识的时序数据库

一些NoSQL时序数据库采用基于标识的数据模型管理时间序列数据，例如InfluxDB[17]、Prometheus[18]等。在这些时序数据库中，时序数据不依赖固定的关系模式，在逻辑上可以直接表示为二维表结构，从而支持基于表的查询与分析。

以InfluxDB为例，其原生模型由测量（measurement）、标识（tag）与字段（field）构成，测量用于界定数据的逻辑分组，标识描述可索引的维度属性，字段则承载具体的度量值。该模型可直接映射为行列结构（时间戳与维度属性形成主键空间，度量值列对应观测变量），从而可直接投射为表视图，如**图3**所示。

这一特性能够在Flux查询语言中执行等价于关系代数的分析操作，而无须经历模式转换和归并过程。尽管InfluxDB的标识模式与表视图之间建立了直接映射，但其本质是表结构的另一种展示形式，与表结构信息相同。基于标识的数据模型管理方法并不属于数据模型转换，无法解决具备层次化路径语义的时序树模型向表结构转换的问题。

时间	地区	城市	设备号	转速	风向
2025-02-28 11:23:59	华北	保定	设备-1	120.0	45.5
……	…	…	…	…	…
2025-02-28 21:23:59	华北	保定	设备-1	120.0	45.5
2025-02-28 11:23:59	华南	汕头	设备-2	160.0	60.5
……	…	…	…	…	…
2025-02-28 21:23:59	华南	汕头	设备-2	160.0	60.5

tag　　field

图3　InfluxDB建模示例

InfluxDB的查询结果以列名为标签，以字段的动态表结构呈现。虽然这种方式更加契合数据分析用户的查询习惯，但由于其缺乏固定结构，用户往往需要对系统内数据信息足够熟悉才能高效使用。

2 树模型的表视图转换

树模型的查询围绕路径展开，常见的操作包括指定时序数据的范围查询、时序数据降采样查询与聚合查询等。然而树模型的路径结构难以支持表视图擅长的多维复杂分析，例如按照设备进行聚合、筛选，以及进行同类设备的对比分析等。

为同时满足工业监控与分析场景的不同需求，本节提出时序表视图的概念和对应转换算法，以实现保留树模型优势下的前提支持复杂分析。

2.1 前置工作

2.1.1 树模型定义

在树模型中，所有元素被抽象为节点（node），节点构成了层级式的树结构。根据在树模型中的语义角色，节点被划分为如下4类，如**图4**所示。

• 根节点（root node）：树结构的起始节点，唯一存在，作为整个数据模型的根。

• 组织节点（organizational node）：用于表示物联网生产环境中的一个物理或逻辑组织，例如区域、工厂或产线。

• 设备节点（device node）：用于表示一个物联网生产环境中的设备，它是多个测点的归属单位。

• 测点节点（measurement node）：用于表示一个实际的数据采集点，对应一个真实存在的传感器，是数据管理的最小单元。

树模型可形式化表示为树结构 $\mathcal{T}=(V,E)$，其中 V 是节点集合，每个节点 $v \in V$ 表示树中的一个层级单位；E 是有向边组成的集合，若 $(v_i, v_j) \in E$，则 v_i 是 v_j 的父节点。更进一步，根据具体语义将树模型中的节点划分为：

$$V=\{v_{\text{root}}\} \cup V_{\text{org}} \cup V_{\text{dev}} \cup V_{\text{meas}} \tag{1}$$

其中，v_{root} 表示全局唯一的根节点；V_{org} 表示组织节点集合，用于组织分层结构；V_{dev} 表示设备节点集合，$\forall v_i \in V_{\text{dev}}$ 均为一个或多个测点节点的父节点；V_{meas} 表示采集时序数据的测点节点组成的集合，$\forall v_j \in V_{\text{meas}}$ 均为叶子节点。

此外，点集合 V 满足如下约束：

• $\forall v \in V \setminus V_{\text{root}}$ 有且仅有一个父节点；

• $\forall v \in V_{\text{dev}}$ 的父节点是组织节点、设备节点或根节点；

• $\forall v \in V_{\text{meas}}$ 是某个设备的子节点。

基于树结构 $\mathcal{T}=(V,E)$，定义节点 $v \in V$ 的路径 $\mathcal{P}(v)$ 如式（2）所示，其满足：

$$\mathcal{P}(v)=(v_{\text{root}}, v_1, \cdots, v_k=v) \tag{2}$$

• 路径起始于根节点 v_{root}；

• $\forall i, (v_i, v_{i+1}) \in E$，即相邻节点满足父子关系；

• $v_k=v$，表示路径的终点为目标节点；

• $\mathcal{P}(v)$ 是 $\mathcal{T}$ 中从根节点到 v 的唯一有向路径（$\mathcal{T}$ 是树结构，该路径唯一存在）。

图4展示了一个电池站下的设备和测点的树状逻辑结构。在该结构中，凡是挂载有测点的节点均被标识为设备节点，而设备节点之间形成了一种层级嵌套的递归结构。也就是说，一条完整的时序路径上可能包含多个设备节点，这些节点按照路径层级自上而下逐级嵌套，形成了一种递归式的设备层级结构。在语义上，每个设备节点仅负责其直接挂载的测点集合，即其路径下最近的测点子节点集合。

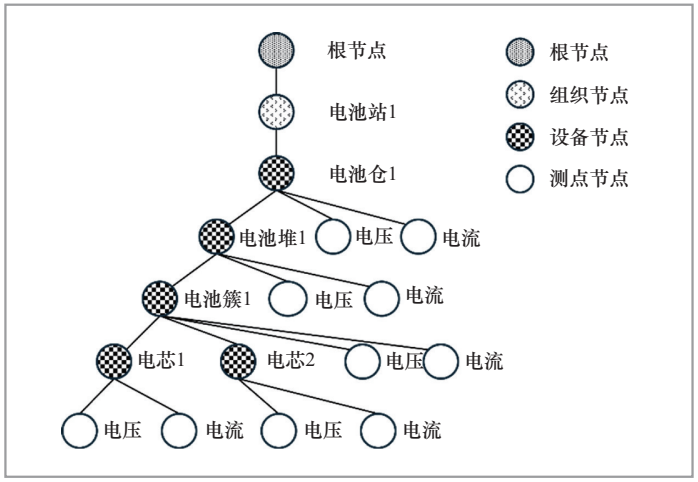


图4 树模型示例

2.1.2 表视图定义

表视图用于管理结构类似的设备，这

些设备通常拥有类似的测点。在表视图中，一行数据表示某个设备的测点在某一时刻的采样信息，因此表视图需要表示设备信息并提取设备测点采样值。根据表视图的语义，其列可被划分为以下3类。

• 时间列：时序表包含一个唯一的时间列Time，其数据类型为时间戳，用于表示离散的采样时间，所有的测点值以时间列为对齐基准。

• 标签列：设表中包含 n 个标签列 $g_1, g_2, \cdots, g_n$，这些列的值具有不可变性，共同构成设备的唯一标识。

• 测点列：设表中包含 m 个测点列 $f_1, f_2, \cdots, f_m$，每个测点列表示一个具体的传感器类型，记录其随时间变化的测量值。

基于上述定义可得表视图模式的形式化定义：

$$\mathcal{S} = (\text{Time}, \mathcal{G}, \mathcal{F}) \tag{3}$$

其中，Time为表视图的唯一时间列；$\mathcal{G} = \{g_1, g_2, \cdots, g_n\}$表示标签列组成的集合；$\mathcal{F} = \{f_1, f_2, \cdots, f_m\}$表示测点列组成的集合。

2.2 树模型的表视图映射

为实现从树模型向表视图的结构映射，需要明确树模型路径中节点在表结构中的语义角色。具体而言，需要解决两个关键问题：

• 如何根据路径结构提取设备的唯一标识，并映射为表视图中的标签列；

• 如何统一不同设备下的测点，使其表示为表视图中的属性列。

2.2.1 表视图映射规则

表视图模式用于定义将树模型转换为结构化二维表的映射规则，需要指定的参数如下。

• 转换范围 v_s：节点 v_s 用于限定表视图覆盖的树模型子树范围，即 v_s 为对应子树的根节点。

• 标识定义 $\vec{\tau}$：向量 $\vec{\tau}$ 用于构造设备标识，其约束了从树路径中提取的标签数量 $n = |\vec{\tau}|$ 及标签对应的语义列名 $\vec{\tau} = (\text{tagname}_1, \cdots, \text{tagname}_n)$；

• 测点集合 $\mathcal{F}$：集合 $\mathcal{F}$ 用于指定表视图需要包含的测点字段，$\mathcal{F} = \{f_1, \cdots, f_k\}$，这些字段是测点路径的后缀，将被搜索并投影为表视图的列。

综上所述，指定转换范围 v_s、标识定义 $\vec{\tau}$ 和测点集合 $\mathcal{F}$ 即可唯一确定一个表视图的模式，其反映了业务关注的设备范围、设备标签语义以及设备测点集合。

2.2.2 设备标识转换

在时序树模型中，每条时间序列路径均标识一个测点，路径层级天然携带了设备测点与上层的结构信息，为此，本文引入了一种基于路径与标签字段的推导机制。该机制允许用户指定一个子树转换范围，从其根后续节点中提取若干字段作为设备的标识信息，从而构建该子树范围下设备的结构化标识信息。

设 v_a 为子树的根节点，v_d 为子树下的设备节点，定义设备节点 v_d 在该子树中可提取的标识的数量为：

$$N_{\text{tag}}(v_d, v_a) = \left|\mathcal{P}(v_d)\right| - \left|\mathcal{P}(v_a)\right| \tag{4}$$

即设备节点 v_d 与子树根点 v_a 在路径中的层级差。

为保证设备标识结构的一致性与标识列数 n 匹配，当且仅当 $N_{\text{tag}}(v_d, v_a) \leq n$ 时，设备 v_d 的路径 $\mathcal{P}(v_d)$ 才会参与表构建，否则将被跳过。

在此基础上，定义设备标识提取函数如式（5）所示。在式（5）中，$\mathrm{tag}(v)$用于提取节点携带的标识信息，故逐一提取位于路径$\mathcal{P}(v_{\mathrm{d}})$但不位于路径$\mathcal{P}(v_{\mathrm{s}})$的所有节点的标识（节点$v_{\mathrm{d}}$是$v_{\mathrm{a}}$的后代）。

$$\Phi(v_{\mathrm{d}}, v_{\mathrm{a}}, n)=\{\mathrm{tag}(v) \mid v \in \mathcal{P}(v_{\mathrm{d}}) \wedge v \notin \mathcal{P}(v_{a}) \wedge N_{\mathrm{tag}}(v_{\mathrm{d}}, v_{\mathrm{a}}) \leqslant n\} \tag{5}$$

设备标识提取函数$\Phi(v_{\mathrm{d}}, v_{\mathrm{a}})$的执行过程如算法1所示。该算法以子树根节点、设备节点与待提取的标签数量为输入，返回一个提取得到的设备标识。对于那些路径过长的设备，即$N_{\mathrm{tag}}(v_{\mathrm{d}}, v_{\mathrm{s}})>n$的设备，算法会跳过对其设备标识的提取，因为有限的标签无法定位到这些设备节点。随后算法会从子树根节点开始，按照标签字段数，逐个提取节点的字面值，填充到标签列表中。

算法1：设备标识提取算法

Input: 子树根节点v_{a}, 设备节点v_{d}, 标签提取字段数n

Output: 标签序列$\Gamma=(\mathrm{tag}_1, \cdots, \mathrm{tag}_n)$

```
if N_tag(v_d, v_a) > n then return
    Γ ← ∅
for each v ∈ P(v_d) ∧ v ∉ P(v_s) do
    Γ.append(tag(v))
while |Γ| < n do
    Γ.append(NULL)
return Γ
```

2.2.3 测点投影

测点投影指的是将业务关心的测点以设备属性的方式进行显式建模。允许用户对指定路径前缀下的设备子树进行扫描与筛选，并将其具有相同测点名称的序列字段投影为表视图中的列。通过测点投影，用户可以构建具有业务含义的“宽表”结构。

$$\Psi_{\mathcal{F}}(v_{\mathrm{d}})=\{v \mid \mathrm{v} \in \mathcal{F} \wedge (v_{\mathrm{d}}, v) \in E\} \tag{6}$$

定义测点投影函数如式（6）所示，其作用于设备节点v_{d}，通过扫描v_{d}的所有测点节点以提取存在于预定义的测点集合$\mathcal{F}$的所有测点。

表视图数据的构建过程如算法2所示。

算法2：表视图的构建算法

Input: 子树根节点v_{s}, 标识定义$\vec{\tau}$, 测点集合$\mathcal{F}$

Output: 结构化表视图的数据集合$\mathcal{D}$

```
1  D ← ∅
2  for each v_d 使得 P(v_d) ∩ P(v_s) = P(v_s) do
3      Γ ← Φ(v_s, v_d, |τ⃗|)
4      if Γ ≠ ∅ then
5          Ω ← Ψ_F(v_d)
6          for each t ∈ time(v_d) do
7              d ← getValues(t, Ω, |F|)
8              δ ← (Γ, t, d)
9              D.append(δ)
10 return D
```

该算法的输入为一个时序树模型的节点、一组标识定义以及一个测点名称集合，首先初始化一个空的结果集，随后遍历所有路径上包含v_{s}的设备节点，并尝试使用算法1从设备节点的路径中提取标签集合，如果发现不为空，则提取该设备下测点的数据。

$\mathrm{getValues}(t, \Omega, |\mathcal{F}|)$表示提取符合测点集合投影的点在$t$时间的值，最终数据集合将上述测点标识集合、时间和测点值集合合并为数据集合的一部分，形成结构化的表视图数据集合。经过设备标识的解析和测点的投影，设备包含在路径中的语义信息被提取为表视图中的标识列，设备的测点

数据被投影为表视图中的测点列数据。

以**图 4** 展示的电池站为例，子树节点划定了业务关心的设备范围，n 的大小则划定了对深层设备的感知能力，如**图 5** 所示。

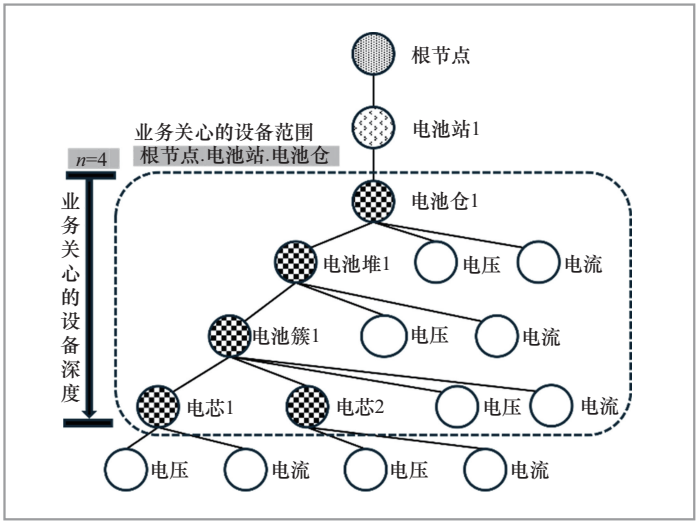


图 5　子树根节点与标识列数 n 对设备提取的影响

2.2.4　表视图映射算法实例

为了方便描述映射方法，本文以**图 4** 所示储能站为例，描述在不同业务下的适用性。**图 4** 所示是一个储能站不同设备的层次结构，储能单位（如电池站、电池仓、电池簇）不仅具有逻辑上的包含关系，其本身也作为设备使用传感器收集信息。

笔者将原来的树结构进行变形，将测点绘制在设备左右两侧，表示设备本身的测点数据，而设备保留位置，因此得到了一个由设备组成的树形结构，如**图 6** 所示。

在一棵时序树模型上，可以依据不同的业务需要进行视图提取。**图 7** 描述了不同的业务采用合适的参数从一棵时序树上提取数据的场景：业务 1 与业务 2 相比，提取测点的层级发生了变化；业务 2 与业务 3 相比，提取的数据范围发生了变化；表（a）与表（b）相比，展示了设备范围提取的效果。

• 业务 1：电池仓 1 的负责人员希望获取管辖范围内所有电池簇的电压以进行安全评估。基于此业务需要，v_a 为“根节点.电池站 1.电池仓 1”，标识定义 $\vec{\tau}$ 为[电池

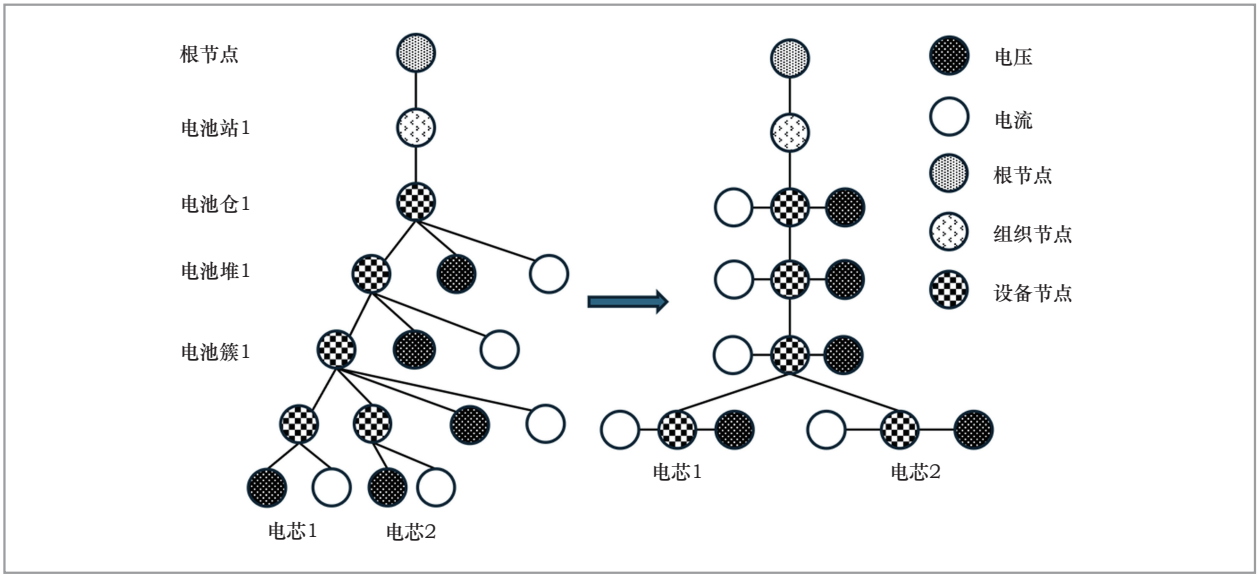


图 6　设备树

堆,电池簇]，即从电池仓1向下提取两级，业务关心的是该节点的电压，因此测点集合 $\mathcal{F}$={电压}，其提取结果如**图7**右表(b)所示，表中显示了电池簇1的两个时间点的电压采样值。

• 业务2：电池仓1的负责人希望获取管辖范围内所有的电池组的电压与电流以计算电池瞬时功率。基于此业务需要，在 v_a 不变的情况下，增加 $\vec{\tau}$ 为[电池堆,电池簇,电池组]，即从电池仓1节点向下提取3级，业务关心的是节点电压与电流，因此测点集合是 $\mathcal{F}$={电压,电流}，其提取结果如**图7**右表(c)所示。

• 业务3：在“电池仓1.电池堆1.电池簇1”中的电池组需要进行实时监控，其余的电池组由于业务范畴的约束无法获得。基于此业务需要，设置 v_a=“根节点.电池站1.电池仓1.电池堆1.电池簇1”，$\vec{\tau}$ = [电池堆]，测点集合为{电压,电流}，其提取结果如**图7**中表(d)所示。与表(c)相比，其关注的提取层级是一样的，区别在于作用范围，表(d)中的数据限制在电池簇1范围内，而表(c)的提取范围则限制在了电池仓1范围。

对于业务跨多个层级提取的场景，例如对电池仓1中的所有设备进行电压监控，可以在提取时允许标识提取空值。在**图7**表(a)描述了这一操作的效果。T_1 时刻，电池堆与电池簇为null对应的是电池仓1的电压，电池堆为电池堆1、电池簇为null对应的电池堆1的电压。归其原因在于，路径前缀与设备标识组合得到了原来设备完整的路径，允许为空路径则意味着组装得到的路径长度是可变的。

2.2.5 模式演化

在时序场景下，数据库模式会随着设备类型、测点配置与业务需求的变化而变化。一方面，新增的设备或者传感器会引入新的设备类型与测点维度，导致模式扩展；另一方面，设备可能因为废弃导致模式收缩。

使用表视图映射可以屏蔽树模型上的模式演化，例如新增设备、新增测点、删除设备等，这些模式演化的结果只会影响视图查询返回标签组合的规模，不会影响视图本身的定义。**图8**描述了新增设备、新增新测点类型与删除旧设备对表视图的影响。新增的设备，若其测点集合符合表

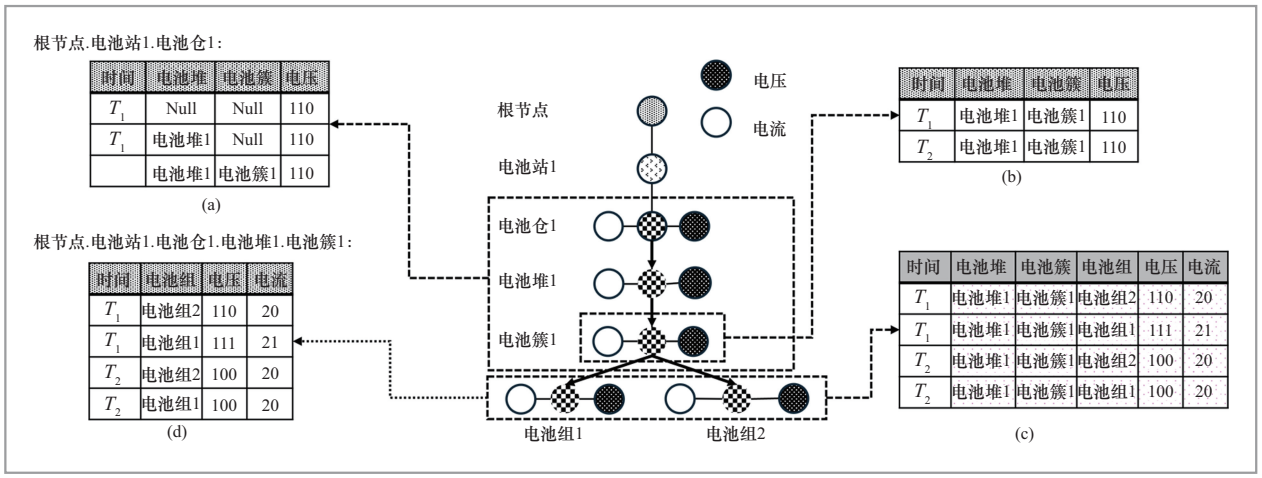


图7 依照业务需求创建的视图示例

视图的电压电流筛选规则，则其测点数据出现在表视图的结果集合中，新增的测点类型不会影响表视图，也不会在查询时被筛选。为了针对新测点类型进行分析，可以创建额外的表视图，无须改动原先的表定义。对于删除的设备电池组2，其测点数据以及标识信息不会在表视图的结果集中出现，例如图8中已经删除的电池组2对应的T_1、T_2、T_3时间戳数据都将消失。

2.3 树模型的表视图查询实现

2.3.1 表视图查询流程

图9展示了面向树模型查询和表视图查询的查询流程和查询任务对象。尽管两种查询在语法表达和语义建模上存在差异，但是它们的执行路径采用了统一的多阶段处理框架。

对于树模型查询，采用专用的时间序列查询语言（time series query language，TSQL）语法，具备对路径结构、层级聚合、通配符等操作的原生支持。语法解析与语义分析阶段负责路径绑定、元数据校验和类型推导，最后基于语义分析结果生成查询执行计划，用于描述查询操作的关系结构与依赖语义。

表视图查询则使用标准的SQL语法，面向关系语义建模，与传统的查询语言兼容。其查询处理过程同样遵循语法解析、语义解析、计划生成等顺序，但在语义分

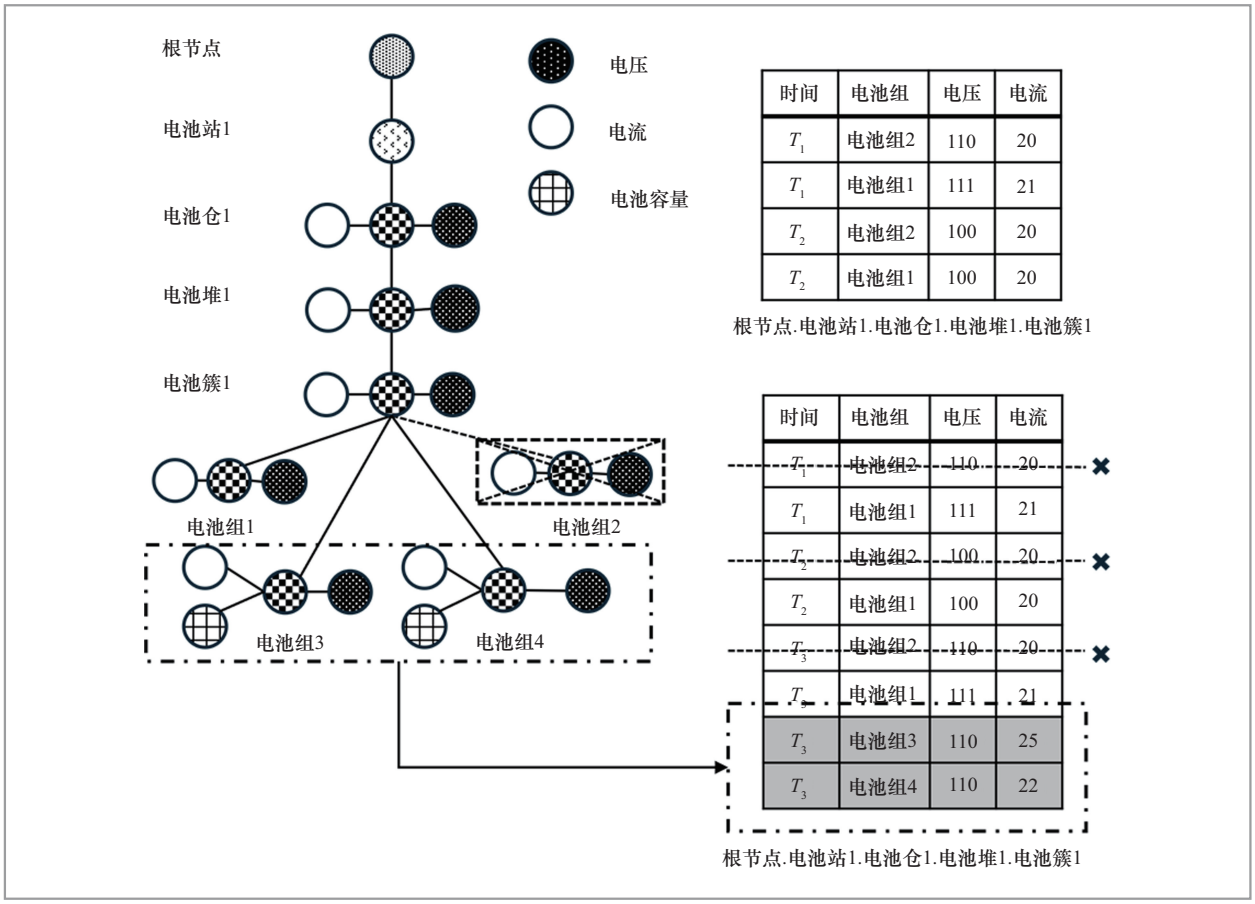


图8 设备与测点类型增减对表视图映射的影响

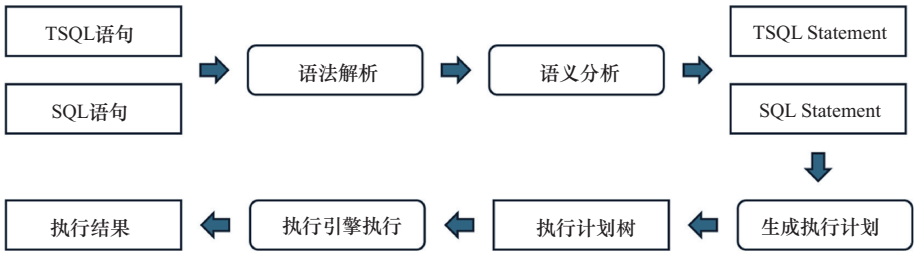


图9 面向树模型查询和表视图查询的处理流程

析阶段重点检查字段与标签映射、字段投影规范性检查等表视图特有的内容。

无论源语法为何模式，都会得到统一的执行计划树，最终由执行引擎执行返回结果。

2.3.2 表视图中设备标识的匹配

在表视图模型下，查询语句包含两类典型过滤条件：一是针对设备标识字段的过滤，二是针对测点属性字段的时间区间与数值过滤。

表视图的标识字段本质上是对树模型路径空间的逻辑映射，这两类过滤条件在查询中分属不同的处理阶段，并被映射为两类不同的执行语义。

针对设备标识字段的过滤实际上在查询计划构造阶段已经发生，并在进行元数据校验时已经通过路径模式（path pattern）完成了对元数据的检索。

路径模式是面向层次结构的检索条件声明模式，IoTDB的树模型与Linux文件系统中的文件模糊匹配机制是典型的应用案例。IoTDB的路径模式支持单层通配符*、单层前缀通配符（例如a*）和多层（1层或者多层）通配符**，典型示例有root.db.d.*、root.db.*.s，以及root.**.s。

IoTDB使用有限自动机（finite automata，FA）[19]进行元数据路径模式匹配。

图10所示为树模型下的路径匹配机制。原始的路径模式经过编译生成FA，根据FA上定义的状态和转移在元数据树上进行检索，其中每个被访问的节点一定能够匹配FA中的某个中间状态，表示最终匹配结果节点一定能够匹配FA中的终止

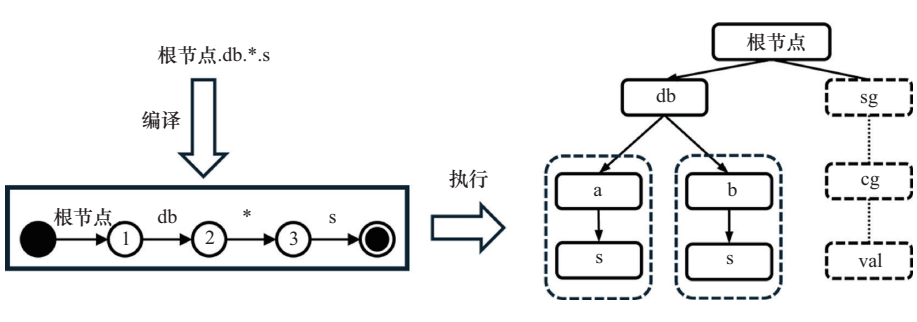


图10 树模型下的路径匹配机制

状态。

在表视图下，将视图对应的路径前缀与标识列的值按照顺序组合后能够得到设备的路径。因此针对设备标识过滤的条件，需要将其转化为对应的路径模式后，再执行元数据检索。

不同复杂程度的标识过滤条件，有不同的处理规则。基于表视图中定义的标识顺序，依次处理相应的标识过滤条件。

• 对于没有指定过滤条件的单个标识列，则对应位置在路径模式中以*或者**填充，进而转换为路径匹配。

• 若指定的条件为前缀过滤，则转换为路径模式中的前缀匹配符。

• 若为等价判断，则转化为路径模式中相应的节点名。

• 针对多个单标识过滤条件的组合，路径模式只能表达and逻辑操作，并不能表达OR逻辑操作，因此可以将逻辑表达式规约成多个以and连接的符合条件OR的连接形式。

• 针对每个符合条件生成对应的路径模式与FA，在FA的基础上进行合并操作以加速过滤。

图11所示为标识过滤条件的解析过程。时序树模型经过表视图映射得到了工厂的设备视图，视图前缀为“根节点.数据库.工厂”，提取标识名为[地区、厂号、设备号]，测点集为{value}。

针对图中包含标识列的查询，谓词条件（地区="华北" AND 厂号="1001"）可以转换为路径匹配：根节点.数据库.工厂.

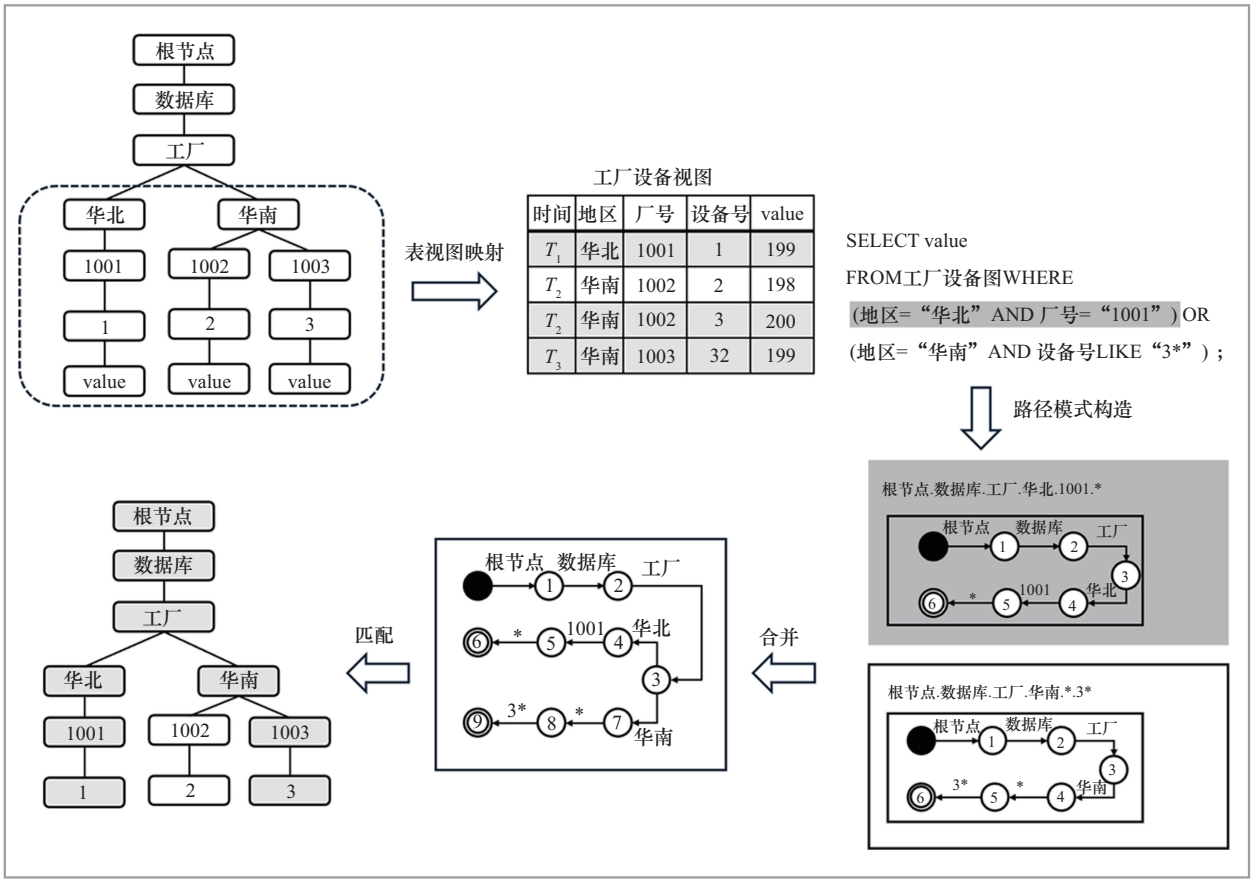


图11 在表视图中进行标识列过滤

华北.1001.*。谓词条件（地区 ="华南" AND 设备号 like "3*"）可以转换为路径匹配：根节点.数据库.工厂.华南.*.3*。为了节省遍历时的成本开销，两个FA将被结合成一个FA后对元数据树进行检索。

2.3.3 表视图中属性列的时间过滤和值过滤

在查询的构造阶段，查询引擎已经获得了待查询的时间序列元数据集合。针对属性的时间过滤查询和值过滤查询可以由树模型的查询机制实现。

2.3.4 表视图中的权限访问控制

在工业场景中，时序数据可能包含敏感信息。为了确保数据的安全性与隐私性，IoTDB通过权限模块实现数据安全和访问控制。

在IoTDB的树模型中，权限系统以子树的粒度控制用户的权限，并将权限按照功能分为READ_DATA、WRITE_DATA、READ_SCHEMA、WRITE_SCHEMA，分别对应树模型中数据和模式的读写权限。

在树模型权限的基础上，表视图的操作也由树模型约束。在创建视图时，校验用户必须具备以下条件：对应数据库的管理权限；对应前缀路径的READ_SCHEMA和READ_DATA权限。在查询视图时，用户则必须具备该视图的查询权限，否则无法读取视图的数据。

3 系统验证与分析

本节使用经典工业物联网数据场景的读写测试用例，对本文提出的表视图算法在写入与查询性能方面的影响进行系统测试。实验主要围绕下面两个核心问题展开分析。

- 表视图的创建是否会对原有树模型的数据写入产生明显影响?
- 基于表视图的查询与树模型的原有查询在查询效率上是否存在差异?

3.1 实验设置

3.1.1 部署环境

测试环境为Ubuntu 22.04操作系统，硬件配置包括Intel(R) Core(TM) i7-10700F处理器，32 GB内存与1.8 TB的HDD磁盘。

3.1.2 测试负载

本次实验使用的数据负载基于TPCx-IoT标准基准测试工具生成。TPCx-IoT由事务处理性能委员会（Transaction Processing Performance Council，TPC）发布，专为评估大规模物联网数据平台的处理能力而设计。该基准以YCSB为基础扩展，模拟典型IoT系统中写入密集型的高并发场景，涵盖大量设备的并发请求场景。

3.1.3 测试方法

测试覆盖写入与查询两类典型任务，针对树模型与表视图进行对比分析。写入测试模拟工业物联网场景中大规模传感器的定时采集与批量上报。测试采用定时批量写入方式，每一批次对所有传感器序列写入相同时间段内的数据。

查询测试则聚焦工业物联网场景在数据管理与分析的实际需求，包括Q1精确点查询、Q2范围查询、Q3聚合查询，以及使用真实场景下的Q4复杂查询与Q5实时

查询。其中，复杂查询来源于1 000 MW超临界锅炉的历史运行日志中，14路内壁与7路外壁的红外同步记录。该查询希望回溯2022年1月1日全天数据，以筛选内壁、外壁温差超过100℃的异常时段。查询结果按时间排序输出，用于离线分析结焦或燃烧偏斜的根本原因。而实时查询则是在100台设备的48路传感器秒级上送环境中，源于运维大屏对于最新采样值的实时查询需求。

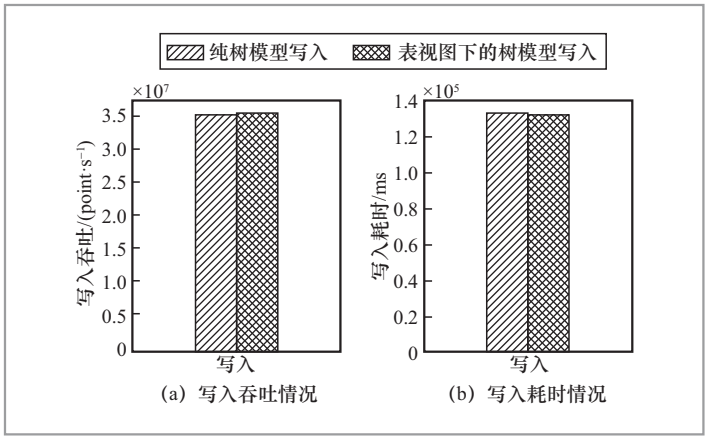


图12 是否使用表视图对写入性能的影响分析

3.1.4 观测指标

为了全面评估各个方案的读写性能，实验过程主要观测以下指标。

- 写入吞吐：指系统每秒钟处理的数据点数量，单位为点每秒（points/s），用于衡量系统对高频数据采集的支撑能力。
- 写入耗时：指单批数据写入需要的时间，单位为ms，反映系统的实时响应能力。
- 平均查询耗时：指各类型查询任务的平均响应时间，单位为ms，衡量系统在复杂查询任务中的性能。
- 元数据创建耗时：指的是创建表视图本身占用的时间，单位为ms，衡量表视图的构建效率。

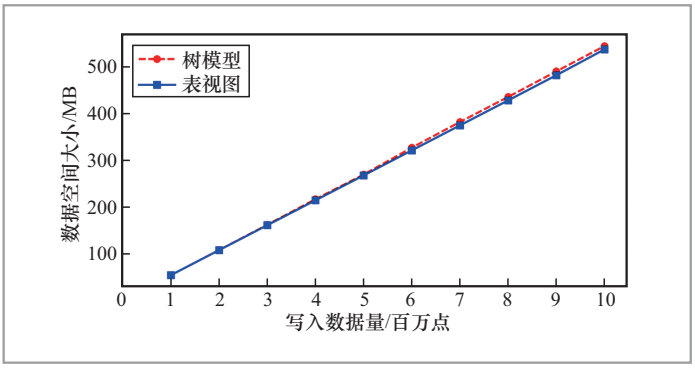


图13 是否使用表视图对于磁盘空间占用的影响

3.2 表视图对写入性能的影响

为了分析表视图创建对写入性能的影响，本次实验在相同的数据集与写入批次划分的前提下，分别执行了纯树模型写入、创建表视图后的树模型写入两种方案，重点观测每次写入的平均写入吞吐、平均写入耗时以及写入占用的磁盘空间。

图12与图13展示了两组对比方案在同一数据集下的写入性能与磁盘空间占用情况。

从图12可以看出，两种方案的平均写入吞吐基本保持在在3.5×10^7左右，差异较小。在写入耗时方面，单批48亿个数据点的写入平均耗时均在130 s左右，这个差距可以认为这两种方案在写入上的表现是基本一致的，说明表视图的创建并没有显著限制写入管道的数据流量。

从图13可知，在写入过程中，写入占用的磁盘空间呈现线性的上涨，说明每次到来批次的数据量基本一致，而采用了表视图的方案，在元数据与数据量上与树模型基本一致，说明表视图的构建过程并不涉及数据的迁移或者复制工作。这个转换逻辑并不会对原有的数据造成影响。

表视图创建对写入性能的影响非常小，这一方面是因为，在IoTDB中，表视图本质上是对已有时间序列在视图上的映射，IoTDB仅需在内存与磁盘的数据上记录一个映射关系，例如路径标识等，并不需要为表结构创建额外的索引或者将数据进行复制迁移。除此之外，IoTDB在执行写入操作时，会先将数据写入内存表并异步刷写到磁盘，而表视图的操作信息则是记录在内部元数据管理模块中。这两个操作的执行路径相对独立，因此表视图的创建不会对数据写入流程有明显影响。

此外，笔者对表视图的平均创建耗时进行了专项测量。在有限次的创建实验当中，单次创建的平均耗时为0.07 s，最长的耗时也未超过0.10 s。这个极小的时间开销在工业级高并发写入场景中基本可以忽略，这也说明了本文提出的表视图的映射机制具有很高的效率，与传统ETL相比，省去了不同数据库间数据模型转换的时间成本。

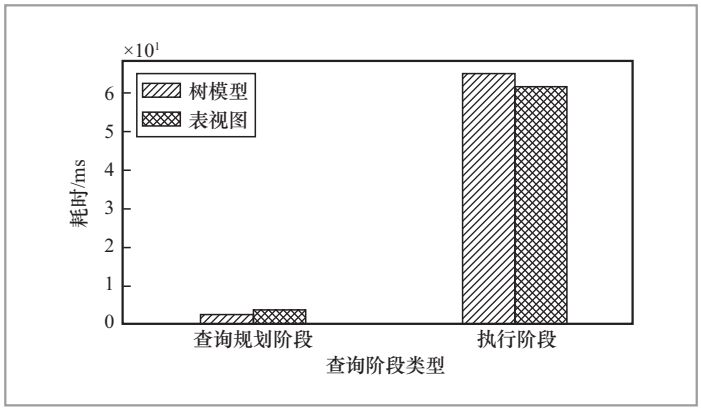


图14 查询阶段对比情况

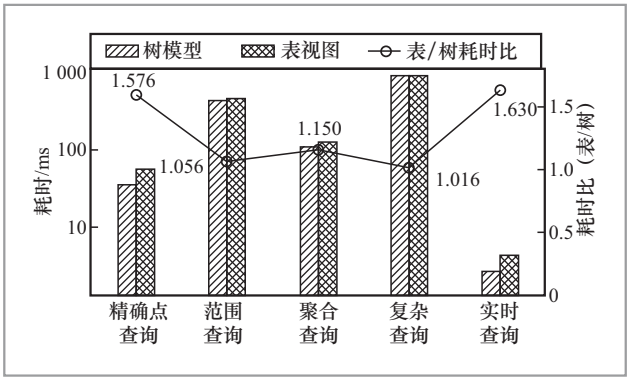


图15 查询任务对比情况

3.3 表视图对查询性能的影响

在查询性能方面，本文将基于树模型的原生查询与基于表视图的查询进行对比。其中主要观测两种方案在查询规划与执行阶段的耗时差异，以及不同查询场景下的整体性能情况。

首先是在查询执行各个阶段的耗时对比。为了能够获取各个阶段的执行耗时，本文使用IoTDB自带的查询计划分析（explain analyze）功能，其可以直接反馈关键阶段的执行时间。**图14**所示为在范围查询下，基于树模型的查询与基于表视图的查询在查询规划阶段与查询执行阶段的耗时差异。

综合来看，尽管表视图的查询在查询规划与物理计划生成过程中引入了少量的额外开销，但是整体对查询执行的影响非常小。

其次是在各种经典查询和实际应用场景下，**图15**所示为在100个设备48个传感器上采集的总计4.8亿个数据点的数据集上，基于树模型的查询与基于表视图的查询在各种查询任务上查询耗时。

从**图15**可以看到，在精确点查询、范围查询与聚合查询等典型场景，以及源自真实业务的复杂任务查询与实时查询中，基于表视图的查询相较于直接在树模型上执行查询，会引入一个较小的耗时增量。该增量主要源于前文描述的元数据映射和转换开销，这个开销是查询准备阶段相关

的固定代价，因此当单次查询本身耗时很短时，该开销在总体的耗时中占比较高，表现得更明显。而当查询涉及较大数据量或者本身计算复杂时，该开销在整体耗时中占比较小，因而总体差异很小。

3.4 表视图随数据增长的可扩展性

表视图的可拓展性是衡量其在实际工业场景中应用价值的关键指标之一，这是因为在真实工业场景中，数据的产生与写入是源源不断的。因此，在这个背景下设计出的表视图，也需要能够在数据量不断增加的情况下保持合理的性能表现。为此，本文考察了表视图的构建耗时与关联查询与复杂聚合耗时随数据量变化的趋势。

如**图16**所示，随着数据不断增长，表视图创建耗时保持稳定水平。这是因为表视图的构建主要是在元数据层进行映射和建立索引结构，核心是解析树模型的结构并生成表结构化的元数据映射关系。映射构建后，数据只需要根据元数据的映射情况，在查询时进行相应的调整即可。因此，表视图构建耗时不会随着数据量的增长而明显增加。这验证了元数据驱动的表视图在大规模数据写入时具有良好的构建稳定性。

相比之下，关联查询的耗时随着数据量的增长而增长，这主要源于其底层的实现逻辑，其采用的关联查询算子是合并关联逻辑，即对有序的时间序列输入，合并关联可以通过线性扫描输入流来完成合并和输出，从而使关联操作的时间复杂度与数据量保持线性关系，即 O(n)的复杂度。因此，当数据量增加时，关联查询需要处理的数据量也相应增长，导致查询耗时呈现与数据量同比上升的趋势。

复杂聚合查询的耗时随着数据量的增长而增长，但其增长趋势相对平缓。聚合函数的处理复杂度主要与数据量有关，在这里使用的聚合函数是平均、求和等O(n)复杂度的操作，且部分过滤条件可提前缩小参与聚合的数据规模，因此总体耗时的增长速率略低于数据量的增长速率。

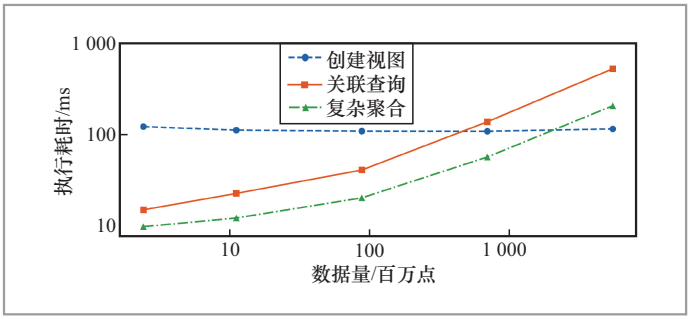


图16 表视图创建、关联查询与复杂聚合的耗时

4 总结与展望

本文提出一种面向统一测点管理与分析场景的融合时序数据建模方法。该方法通过引入表视图机制，在保留树模型灵活组织与高效采集能力的同时，实现具备关系语义的分析接口，为时序数据的统一管理与多维分析提供了系统支持。

为了验证表视图机制的可行性，本文基于IoTDB实现了表视图的转换算法和对应的查询系统，并使用标准基准测试进行性能评估，结果发现，其在写入与查询性能方面对原系统无明显负担，具备较好的实用性与工程可行性。

本文表明，通过合理设计数据模型转换机制，可以有效打通监控与分析之间的数据通道，避免重复存储与复杂ETL，降低系统复杂性，提升数据处理效率。

对于模式转换的研究，未来的工作可以从以下4个方面展开。

• 探索基于统一执行引擎的树、表视图协同优化策略，提升跨模型分析的性能。

• 针对标识列的过滤是将过滤条件转化为路径匹配规则处理的，在复杂过滤条件和大规模时间序列的条件下可能会有性能问题，可以探索跨模型的联合索引机制，加速标识列的过滤。

• 本文依赖用户定义视图模式，匹配算法简单，为了统计不同层级的设备信息，设备标识列很宽，可探索考虑元数据树特征的提取算法，减少视图的大小与数量。

• 引入表视图后需要考虑数据源的权限问题，可设计基于视图细粒度的权限管理机制，提升系统的数据安全性和多租户适用能力。

参考文献：

[1] Xu l d, he w, li S C. Internet of Things in industries: a survey[J]. IEEE Transactions on Industrial Informatics, 2014, 10(4): 2233-2243.

[2] 刘帅，乔颖，罗雄飞，等. 时序数据库关键技术综述[J]. 计算机研究与发展，2024，61(3)：614-638.
LIU S, QIAO Y, LUO X F, et al. Key techniques of time series databases: a survey[J]. Journal of Computer Research and Development, 2024, 61(3): 614-638.

[3] 王建民. 工业大数据技术综述[J]. 大数据，2017，3(6)：3-14.
WANG J M. Survey on industrial big data [J]. Big Data Research, 2017, 3(6): 3-14.

[4] 周惠平. KKS编码系统在电厂设备基础信息库的应用分析[J]. 机电工程技术，2014，43(8)：79-81，106.
ZHOU H P. Application analysis of KKS coding system in equipment foundation database of power plants[J]. Mechanical & Electrical Engineering Technology, 2014, 43(8): 79-81, 106.

[5] TUBAISHAT M, YIN J, PANJA B, et al. A secure hierarchical model for sensor network[J]. ACM SIGMOD Record, 2004, 33(1): 7-13.

[6] CHEN J, HE J Y, CHEN F F, et al. Towards general industrial intelligence: a survey of continual large models in industrial IoT[EB]. arXiv preprint, 2024, arXiv: 2409.01207.

[7] CODD E F. A relational model of data for large shared data banks[J]. Communications of the ACM, 1970, 13(6): 377-387.

[8] NAMIOT D. Time series databases[J]. DAMDID/RCDL, 2015, 1536: 132-137.

[9] GARDNER S R. Building the data warehouse[J]. Communications of the ACM, 1998, 41(9): 52-60.

[10] WANG C, QIAO J L, HUANG X D, et al. Apache IoTDB: a time series database for large scale IoT applications[J]. ACM Transactions on Database Systems, 2025, 50(2): 1-45.

[11] KROENKE D M, AUER D J, VANDENBERG S L, et al. Database concepts[M]. Upper Saddle River: Prentice Hall, 2010.

[12] WANG C, HUANG X D, QIAO J L, et al. Apache IoTDB: time-series database for Internet of Things[J]. Proceedings of the VLDB Endowment, 2020, 13(12): 2901-2904.

[13] RODDICK J F. Schema evolution in database systems: an annotated bibliography[J]. ACM SIGMOD Record, 1992, 21(4): 35-40.

[14] 赵鑫，万英格，刘英博. 一种时序数据模式演化的跟踪与查询方法[J]. 计算机研究与发展，2022，59(9)：1869-1886.

ZHAO X, WAN Y G, LIU Y B. Tracking and querying over timeseries data with schema evolution[J]. Journal of Computer Research and Development, 2022, 59(9): 1869-1886.

[15] STEFANCOVA E. Evaluation of the TimescaleDB PostgreSQL time series extension[R]. 2018.

[16] MOMJIAN B. PostgreSQL: Introduction and Concepts[M]. Harlow: Addison-Wesley, 2001.

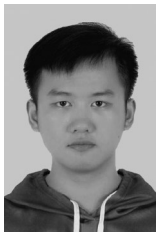
[17] NAQVI S N Z, YFANTIDOU S, ZIMÁNYI E. Time series databases and influxdb[J]. Studienarbeit, Université Libre de Bruxelles, 2017, 12: 1-44.

[18] TURNBULL J. Monitoring with Prometheus[M]. Turnbull Press, 2018.

[19] HOPCROFT J E, MOTWANI R, ULLMAN J D. Introduction to Automata Theory, Languages, and Computation [M]. 3rd ed. Boston: Pearson Addison-Wesley, 2007.

作者简介

李烁麟（1998– ），男，清华大学软件学院硕士生，主要研究方向为工业时序数据库。



林欣涛（2001– ），男，清华大学软件学院硕士生，主要研究方向为工业时序数据库。

田原（1997– ），男，天谋科技（上海）有限公司高级工程师，主要研究方向为工业时序数据库。

许京奕（1973– ），男，博士，北京大数据先进技术研究院正高级工程师、工程中心主任，主要研究方向为人工智能。

陈荣钊（2000– ）男，天谋科技（北京）有限公司高级工程师，主要研究方向为工业时序数据库。

李永瑾（1982– ），女，北京大数据先进技术研究院高级工程师、工程中心副主任，主要研究方向为人工智能。

乔嘉林（1993– ），男，博士，天谋科技（北京）有限公司高级工程师、首席技术官，主要研究方向为工业时序数据库。

收稿日期: 2025–09–22
基金项目: 国家自然科学基金项目(No.62021002)
Foundation Item: The National Natural Science Foundation of China(No.62021002)