软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(3):1154−1172 [doi: 10.13328/j.cnki.jos.007070]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

面向云边端协同的多模态数据建模技术及其应用*

崔双双, 吴 限, 王宏志, 吴 昊

(哈尔滨工业大学 计算学部, 黑龙江 哈尔滨 150001)

通信作者: 王宏志, Email: wangzh@hit.edu.cn

摘 要: 云边端协同架构中数据类型多样, 各级存储资源与计算资源存在差异, 给数据管理带来新的挑战. 现有数据模型或者数据模型的简单叠加, 都难以同时满足云边端中多模态数据管理和协同管理需求. 因此, 研究面向云边端协同的多模态数据建模技术成为重要问题. 其核心在于, 如何高效地从云边端三层架构中得到满足应用所需的查询结果. 从云边端三层数据的数据类型出发, 提出了面向云边端协同的多模态数据建模技术, 给出了基于元组的多模态数据模型定义, 设计了 6 种基类, 解决多模态数据统一表征困难的问题; 提出了云边端协同查询的基本数据操作体系, 以满足云边端业务场景的查询需求; 给出了多模态数据模型的完整性约束, 为查询优化奠定了理论基础. 最后, 给出了面向云边端协同多模态数据模型的示范应用, 并从数据存储时间、存储空间和查询时间这 3 个方面对所提出的数据模型存储方法进行了验证. 实验结果表明, 所提方案能够有效地表示云边端协同架构中的多模态数据.

关键词: 多模态数据模型; 云边端协同; 查询处理

中图法分类号: TP311

中文引用格式: 崔双双, 吴限, 王宏志, 吴昊. 面向云边端协同的多模态数据建模技术及其应用. 软件学报, 2024, 35(3): 1154–1172. http://www.jos.org.cn/1000-9825/7070.htm

英文引用格式: Cui SS, Wu X, Wang HZ, Wu H. Multimodal Data Modeling Technology and Its application for Cloud-edge-device Collaboration. Ruan Jian Xue Bao/Journal of Software, 2024, 35(3): 1154–1172 (in Chinese). http://www.jos.org.cn/1000-9825/7070.htm

Multimodal Data Modeling Technology and Its application for Cloud-edge-device Collaboration

CUI Shuang-Shuang, WU Xian, WANG Hong-Zhi, WU Hao

(Faculty of Computing, Harbin Institute of Technology, Harbin 150001, China)

Abstract: In the cloud-edge-device collaborative architecture, data types are diverse, and there are differences in storage resources and computing resources at all levels, which bring new challenges to data management. The existing data models or simple superposition of data models are difficult to meet the requirements of multimodal data management and collaborative management in the cloud-edge-device. Therefore, research on multimodal data modeling technology for cloud-edge-device collaboration has become an important issue. The core is how to efficiently obtain the query results that meet the needs of the application from the three-tier architecture of cloud-edge-device. Starting from the data types of the three-layer data of cloud-edge-device, this study proposes a multimodal data modeling technology for cloud-edge-device collaboration, gives the definition of multimodal data model based on tuples, and designs six base classes to solve the problem of unified representation of multimodal data. The basic data operation architecture of cloud-edge-device collaborative query is also proposed to meet the query requirements of cloud-edge-device business scenarios. The integrity constraints of the multimodal data model are given, which lays a theoretical foundation for query optimization. Finally, a

* 基金项目: 国家自然科学基金(62232005, U1866602); 国家重点研发计划(2021YFB3300502)
本文由“面向多模态数据的新型数据库技术”专题特约编辑彭智勇教授、高云君教授、李国良教授、许建秋教授推荐.
收稿时间: 2023-07-16; 修改时间: 2023-09-05; 采用时间: 2023-10-24; jos 在线出版时间: 2023-11-08
CNKI 网络首发时间: 2023-12-28

demonstration application of the multimodal data model for cloud edge-device collaboration was given, and the proposed data model storage method was verified from three aspects of data storage time, storage space and query time. The experimental results show that the proposed scheme can effectively represent the multimodal data in the cloud-edge-device collaborative architecture.
Key words: multimodal data model; cloud-edge-device collaboration; query processing

当前, 云边端协同的新计算模式通过整合云计算与边缘计算优势, 已经成为国家数字经济发展的重要战略. “协同发展云服务与边缘计算服务”被列入国家十四五发展规划. IDC 指出: 预计到 2025 年, 将有超过 50% 的数据需要在边缘侧进行存储、分析、计算[1]. 构筑云边端数据存储、通信、处理能力全面协同的数据管理平台迫在眉睫. 然而, 如何构建面向云边端协同的数据高效管理平台仍然存在诸多问题: 首先, 云边端协同架构中数据类型和数据格式呈现多种样式, 需要设计能够统一表征多种不同类型数据的数据模型; 其次, 云边端协同架构具有异构性, 在数据规模、分布、模式和设备的计算能力都存在差异的情况下, 需要对资源进行合理分配与协调管理; 最后, 云边端协同架构中, 海量终端设备以较高的频率采集数据, 数据规模逐渐扩大, 数据管理技术需具备面向大规模数据查询的实时响应能力. 因此, 对于构建云边端协同的数据管理平台, 最基础且最具挑战性的问题之一是如何对多模态数据进行建模, 以实现高效地从云边端三层架构中得到满足应用所需的查询结果.

通过分析云边端协同架构中的数据特点以及相关工作, 本文总结出面向云边端协同的多模态数据建模技术存在的 3 点挑战: 第一, 云边端架构中数据类型多样, 现有数据模型或者数据模型的简单叠加难以支撑多模态数据管理; 第二, 云边端架构中设备和资源异构, 单侧、两侧协同以及三侧协同对数据模型的需求均不同; 第三, 面向云边端协同的数据操作复杂多样, 传统代数运算难以直接支撑复杂的协同数据处理, 而多种数据操作的简单组合嵌套又会降低数据操作执行的效率.

为此, 本文提出了面向云边端协同的多模态数据建模技术. 从设计表征云边端协同多模态数据的数据结构出发, 然后对云边端协同查询的多模态数据基本数据操作体系进行定义, 接着对多模态数据模型的完整性约束进行研究, 最后讨论云边端协同多模态数据模型的示范应用并进行实验验证.

本文的主要贡献如下.

(1) 提出一种基于元组的面向云边端协同的多模态数据模型, 本模型具有 6 种“基类”, 既可以描述云边端单侧的多模态数据, 又能描述云边端三层多模态数据的协同依赖关系.

(2) 提出了面向云边端协同的多模态数据操作体系, 包括查询操作和插入、删除、修改操作, 既满足云边端独立查询操作以及云边端协同查询操作, 并且具有一定的操作扩展性, 以支持用户对数据库进行操作扩展, 为查询处理与优化奠定基础.

(3) 对多模态数据模型的完整性约束进行了研究, 主要包括主键约束、外键约束、范式与事务及其 ACID.

(4) 以智能工厂为例, 介绍了多模态数据模型的应用; 并从多模态数据存储时间、存储空间和查询时间这 3 个方面对所提出的数据模型存储方法进行了验证.

第 1 节介绍多模态数据模型的相关工作. 第 2 节介绍本文提出的面向云边端协同的多模态数据模型. 第 3 节介绍本文提出的面向云边端协同的多模态数据操作体系. 第 4 节主要介绍多模态数据模型的完整性约束. 第 5 节主要介绍面向智能工厂的多模态数据模型应用示例. 第 6 节通过对比实验对提出的多模态数据模型存储方法进行分析验证. 最后总结全文并展望未来工作.

1 相关工作

近年来, 多模态数据是一个比较热门的研究方向, 但是针对多模态数据的研究大多集中在机器学习尤其是深度学习方面, 主要目标是将多模态数据进行融合[2], 通过集成不同类型的特征来提高机器学习模型的性能, 并没有对多模态数据本身的存取进行设计[3]. 其中, 部分需要对数据进行预处理的研究也只是设计了一些简单的数据结构, 而没有涉及多模态数据的数据操作和完整性约束等理论研究.

而在数据库理论研究中, 也鲜有对多模态数据模型的完整研究. 在数据管理理论中, 最经典的数据模型

是关系数据模型[4], 包括基础关系数据结构、基础关系数据操作、关系代数和关系数据完整性约束理论. 随着多类型数据的出现, 多种非关系型数据模型被提出, 其中比较有代表性的是 XML 数据模型[5]、JSON 数据模型、RDF 数据模型、Property 图模型[6]等. 其中, XML 数据模型明确地规范了 XML 文档基础模型、文档数据操作及 XML 代数操作、XML 数据完整性约束, 提高了文档数据的存储和查询效率. Property 图模型[6]定义了基础的属性图数据结构以及属性图数据操作, 为多对多复杂实体存储和查询提供了解决方案. 但是, 传统的为单模态设计的数据模型在面对多模态数据时, 往往会产生结构不兼容、数据丢失、数据冗余、破坏完整性约束等一系列问题.

随着数据中心和数据湖的发展, 某一个数据业务不再仅与单一模态数据相关. 为此, 研究人员对单模态数据模型进行了拓展以支持多模态数据, 如 SQL-extend、XML-extend[7]等. extend 能够利用数据库已有的数据类型创建新的数据类型, 或使用已有的视图创建新的视图. 这些新的数据类型和视图可以继承父类型的特性, 并且允许添加额外的属性和方法, 使数据的组织和存储更加灵活. 尽管 extend 可以建立较为复杂的数据模型, 但是面对复杂多样的云边端协同多模态数据操作场景, 由于 SQL-extend, XML-extend 缺乏对数据完整性约束和代数理论的研究, 难以支撑云边端协同查询处理. 为了满足云边端协同处理的需求, 本文提出了完整的多模态数据模型及其理论体系.

2 面向云边端协同的多模态数据模型

云边端协同架构中的数据按照存储位置能够分为 3 类——端侧数据、边侧数据以及云侧数据, 并且各侧存储的数据规模存在差异, 各侧存储数据的数据类型通常包括关系型数据、图数据、图像、视频、时序数据以及流数据等. 为了统一表征云边端架构中的多模态数据, 本文提出了云-边-端协同架构多模态数据模型, 如图 1 所示.

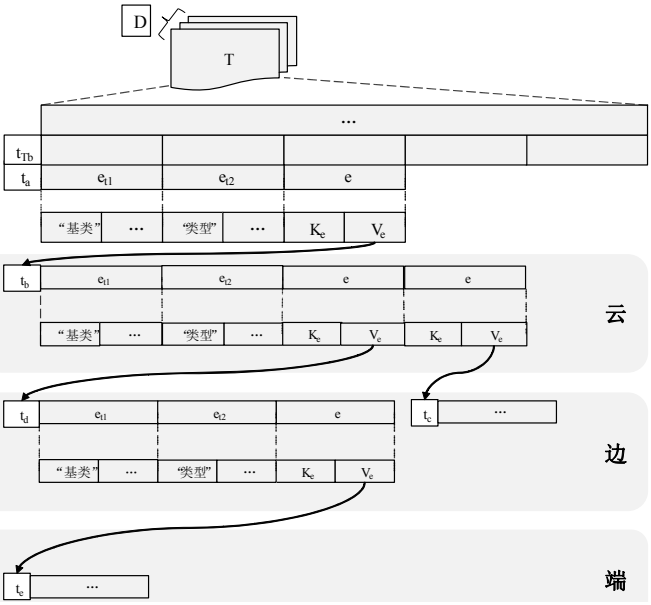


图 1 云-边-端协同架构多模态数据模型示意图

以元组为基础设计多模态数据模型, 并设计了 6 种“基类”, 包括点基类、线基类、属性基类、高维时序基类、时序基类以及编码基类. 其中: 点基类作为最基础的基类数据, 包括连线属性, 能够表示关系型数据与图数据中的结点数据; 线基类能够表示图数据中的边数据; 属性基类可以对关系型数据、图数据中的结点与边数据进行扩展; 高维时序基类能够表示高维度的时序数据; 时序数据能够表示时间序列数据; 编码基类能够

表示图像、视频、流数据等数据. 此数据模型满足了多模态数据表征需求, 既能够描述云边端单侧的多模态数据, 又能描述云边端三层多模态数据的协同依赖关系. 下面对一些构建多模态数据模型用到的定义给出形式化 描述.

定义 1(元素). 元素是二元组 $e=(K_e,V_e)$, 其中, $K_e\in key$, $key=num\cup string$, $V_e\in val$, $val=num\cup string\cup quantities\cup address\cup code$. 元素的取值范围记为 dom. 前者称为元素的“键”, 后者称为元素的“值”.

定义 2(元组). 元组是元素的有序集合, 记为 t, $t\in dom^*$, $dom^*=dom\cup dom\times dom\cup dom\times dom\times dom\cup\ldots$. 当有 n 个元素时, t 的其中第 i 个元素记为 e_{t_i}, 对 $\forall j,k\in[1,n], j\neq k$, 有 $K_{e_{tj}}\neq K_{e_{tk}}$. 特别地, $K_{e_{t1}}$=“基类”, $K_{e_{t2}}$=“类型”, $V_{e_{t1}}\in foun$.

例如, 图 1 中的每一行可以看作一个元组, 每个元组中, 元素的键的取值不可重复. 每个元组的第 1 个元素必须是关于“基类”的键值对, 第 2 个元素必须是关于“类型”的键值对.

定义 3(表). 表是相同基类元组的集合, 记为 T, $\forall t_{T_j},t_{T_k}\in T$, 有 $e_{t_{T_j}}=e_{t_{T_k}}$. 把云边端三层表分别记为 T_{Cloud}, T_{Edge} 和 T_{Device}.

定义 4(数据库). 数据库是不同基类表的集合, 记为 D. 对 $\forall T_j,T_k\in D$, $T_j\neq T_k$, 有 $f_{T_j}\neq f_{T_k}$.

接下来, 根据不同类型数据的表征需求, 将多模态数据模型细分为基于属性基类的云边端协同架构关系型数据模型、基于点基类与线基类的云边端协同架构图数据模型和基于时序基类与高维时序基类的云边端协同架构时序数据模型, 并对每种数据建模技术进行详细介绍.

2.1 基于属性基类的云边端协同架构关系型数据模型

在云边端协同架构中, 当存储关系型数据时, 其属性数据是从端-边-云这 3 层逐级汇聚的. 因此, 表征关系型数据的核心在于解决各侧属性之间的迭代关系. 本文提出的多模态数据模型可以通过属性基类进行扩展, 基于属性基类的数据模型的层数迭代示意图如图 2 所示.

图 2 基于属性基类的数据模型示意图

定义 5(属性基类元组). 以“属性”作为基类的元组不用必须含有某个特定键的元素, 但必须拥有除了“基类”和“类型”以外的至少 1 个元素, 这些元素的“值”往往是任意实数、字符串、物理量、编码和指向其他属性基类元组的地址, 这种元组被称为属性基类元组.

基于属性基类的关系数据建模实例如图 3 所示. 类似地, 云侧数据元素的值中可以存储边侧、端侧数据.

同样地, 边侧数据元素的值中也可以存储端侧数据.

学生信息表

Sno	Sname	Ssex	Sage
001	Nick	M	20
002	Elsa	F	19
003	Ed	M	18
004	Cindy	F	19

基类	类型	Sno	Sname	Ssex	Sage
属性	学生	001	Nick	M	20
属性	学生	002	Elsa	F	19
属性	学生	003	Ed	M	18
属性	学生	004	Cindy	F	19

图 3 基于属性基类的关系数据建模实例

2.2 基于点基类与线基类的云边端协同架构图数据模型

在云边端协同架构中, 当存储图数据, 特别是有向图时, 需要表征数据之间的关联. 为此, 本文提出了“点基类”和“线基类”来表征图数据, 基于点基类与线基类表征图数据示意图如图 4 所示.

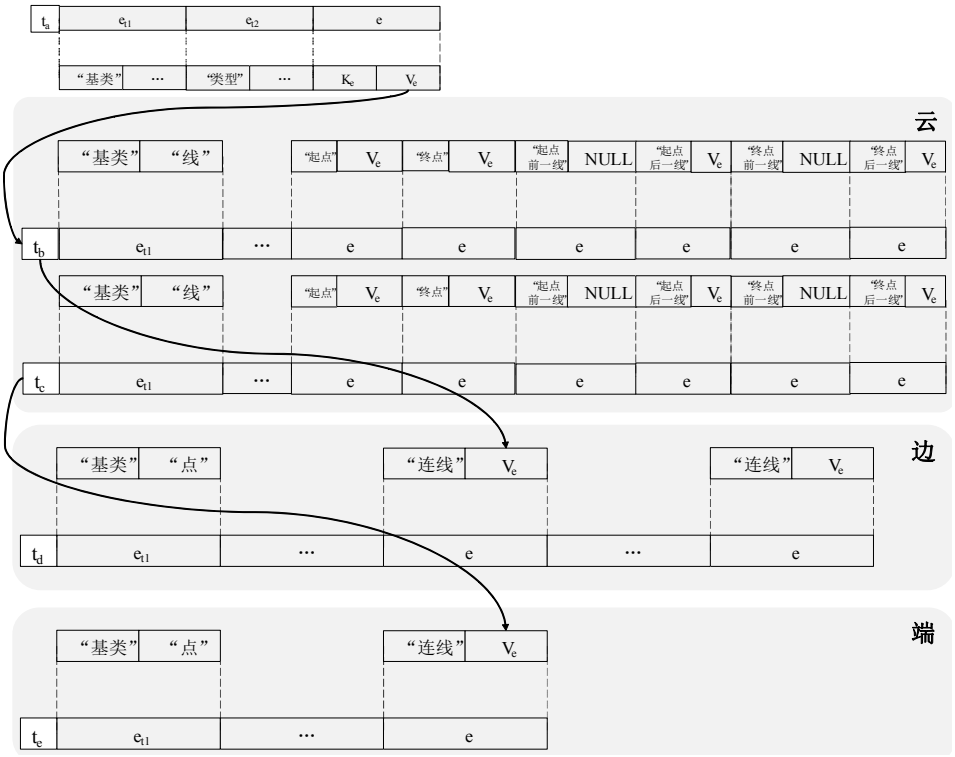


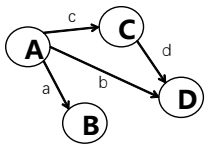
图 4 基于点基类与线基类的数据模型示意图

定义 6(点基类元组). 以“点”作为基类的元组必须含有以“连线”为“键”的元素, 这种元组被称为点基类元组. 对于元组 t, 如果 f_t=“点”, 则$\exists e\in t$, 使得 K_e=“连线”.

定义 7(线基类元组). 以“线”作为基类的元组必须含有以“起点”“终点”“起点前一线”“起点后一线”“终点前一线”“终点后一线”为“键”的 6 个元素, 这种元组被称为线基类元组. 对于元组 t, 如果 f_t=“线”, 则$\exists e_1, e_2,e_3,e_4,e_5,e_6\in t$, 使得 K_{e_1}=“起点”, K_{e_2}=“终点”, K_{e_3}=“起点前一线”, K_{e_4}=“起点后一线”, K_{e_5}=“终点前一线”, K_{e_6}=“终点后一线”.

同样地, 这些元素的“值”均可以为空, 这种灵活性在图结构中是必要的. 在有向图结构的查询中, 从单点出发查询其所有的边是一种常见的业务. 在复杂的查询中, 为了避免对结点数据的频繁访问, 防止每一次对结点数据的访问都从起点开始遍历一个点的边, 设计了“起点前一线”和“终点后一线”两个键, 这可以对查询

带来大幅度的优化. 基于点基类与线基类的图数据建模实例如图 5 所示.



基类	类型	名称	连线1	连线2	连线3
点	X	A	a	b	c
点	X	B	a	NULL	NULL
点	X	C	c	d	NULL
点	X	D	b	d	NULL

基类	类型	名称	起点	终点	起点前一线	起点后一线	终点前一线	终点后一线
线	Y	a	A	B	NULL	c	NULL	NULL
线	Y	b	A	D	NULL	a	d	NULL
线	Y	c	A	C	NULL	b	NULL	d
线	Y	d	C	D	c	NULL	b	NULL

图 5 基于点基类与线基类的图数据建模实例

2.3 基于时序基类与高维时序基类的云边端协同架构时序数据模型

在云边端协同架构中, 传感器采集的时序数据通常规模大、维度高. 为满足时序数据的存储特点, 将时序数据的维度关系视为树结构, 从某个“高维时序”的元组作为根节点出发, 其元素的值可以指向其他“高维时序”的元组; 将其视为非叶节点, 也可以指向“时序”的元组; 将其视为叶节点, 基于时序基类与高维时序基类的数据模型示意图如图 6 所示.

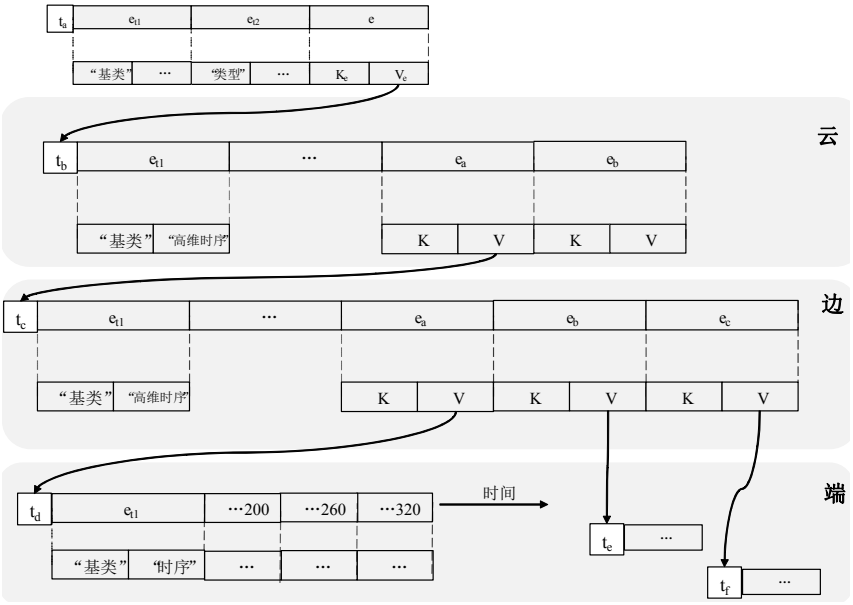


图 6 基于时序基类与高维时序基类的数据模型示意图

定义 8(时序基类元组). 以“时序”作为基类的元组没有必须含有某个特定键的元素, 但必须拥有除了“基类”和“类型”以外的至少 1 个元素, 这些元素以时间戳为“键”, 往往以任意实数、物理量为“值”, 表达该值随时间的变化, 这种元组被称为时序基类元组.

定义 9(高维时序基类元组). 以“高维时序”作为基类的元组没有必须含有某个特定键的元素, 但必须拥有除了“基类”和“类型”以外的至少 1 个元素, 这些元素的“值”是指向其他高维时序基类元组的地址或者是指向时序基类元组的地址, 这种元组被称为高维时序基类元组.

基于时序基类与高维时序基类的时序数据建模实例如图 7 所示. 类似地, 云侧与边侧数据可以采用高维时序基类数据来表征, 端侧数据可以通过时序基类数据来表征.

X 时序信息

Timestamp	Value
Timestamp 1	value
Timestamp 2	Value
Timestamp 3	value
Timestamp 4	value

Y 时序信息

Timestamp	Value
Timestamp 1	value
Timestamp 2	Value
Timestamp 3	value
Timestamp 4	value

Z 时序信息

Timestamp	Value
Timestamp 1	value
Timestamp 2	Value
Timestamp 3	value
Timestamp 4	value

基类	类型	Timestamp 1	Timestamp 2	Timestamp 3	Timestamp 4
时序	X	value	value	value	value
时序	Y	value	value	value	value
时序	Z	value	value	value	value

基类	类型	X	Y	Z
高维时序	A	(Ts, value)	(Ts, value)	(Ts, value)
高维时序	A	(Ts, value)	(Ts, value)	(Ts, value)
高维时序	A	(Ts, value)	(Ts, value)	(Ts, value)

图 7 基于时序基类与高维时序基类的时序数据建模实例

3 面向云边端协同的多模态数据操作

与传统数据库数据操作相比, 面向云边端协同的多模态数据操作复杂多样, 不仅要支撑云侧的多模态数据基础范围过滤操作以及多模态数据聚集过滤等操作、边侧的多模态数据基础过滤操作、端侧面向采集多模态数据的 ad-hoc 和连续查询等操作, 还面临着边-边协同、端-端协同、云-边协同、边-端协同、云-边-端协同场景的复杂查询需求. 本文提出的面向云边端协同的多模态数据操作包括查询操作和插入、删除、修改操作两大部分. 接下来, 首先在第 3.1 节中给出多模态数据查询操作的相关数学定义; 然后, 在第 3.2 节中介绍面向云边端协同的多模态数据查询操作; 最后, 在第 3.3 节中介绍面向云边端协同的多模态数据变化操作, 包括插入、删除、修改操作.

3.1 多模态数据查询操作数学定义

为了更好地理解多模态数据查询操作的数学定义, 首先给出一个多模态数据查询操作用例.

- 多模态数据查询操作用例

“*A*, *B* 均为点基类数据或属性基类数据, *C* 为线基类数据. 为查找所有 *A* 与 *B* 的组合, 满足以下条件: *A*, *B* 和 *C* 为 3 个不同的物体 *M*, *A* 与 *B* 之间有一个 *C*, *CA* 之间、*CB* 之间均通过物体 *N* 进行连接, 这些 *N* 的编号需要小于等于 3, 以 *C* 为起点, *A* 的长度小于 *B* 的长度或者 *A* 的长度等于 *B* 的长度.”

将其转化成类似于 Cypher 的语句[8].

RETURN (*A*,*B*) *MATCH* (*A*)←[*a*]-(*C*)-[*b*]→(*B*) **WHERE** *A*[类型]=“*M*”, *B*[类型]=“*M*”, *C*[类型]=“*M*”, (*A*[长度]<*B*[长度])∨(*A*[长度]=*B*[长度]), *a*[类型]=“*N*”, *b*[类型]=“*N*”, *a*[编号]≤3, *b*[编号]≤3.

该查询语句主要包括以下 3 个要素, 称作查询任务的“对象”“模式”和“条件”.

- 查询的对象“(*A*,*B*)”, 其作为一个二元组, 理应是点集的二阶笛卡尔积的子集, 查询操作需要做到的就是找到这个子集.
- 查询对象的模式“(*A*)←[*a*]-(*C*)-[*b*]→(*B*)”, 含义为 *C* 到 *A* 和 *B* 分别有名为 *a* 和 *b* 的两个有向边, 这部分表示了查询任务中需要找到的那个子集需要满足的模式条件.
- 查询对象的条件“*A*[类型]=“*M*”, *B*[类型]=“*M*”, *C*[类型]=“*M*”, (*A*[长度]<*B*[长度])∨(*A*[长度]=*B*[长度]), *a*[类型]=“*N*”, *b*[类型]=“*N*”, *a*[编号]≤3, *b*[编号]≤3”, 这部分表示了前一部分模式中出现的变量所需要满足的条件, 这些条件的对象既包括被查询的二元组里面的要素, 也包括模式里面出现的其他变量的要素; 而这些条件的内容既包括这些要素本身的属性, 也包括要素与要素之间属性的关系.

下面给出查询操作的数学定义.

定义 10(模式变量). 模式变量指出现在查询任务中模式部分的要素, 视为变量, 记为 ve, 根据模式中的位置分为点和线两类, 记为 vep 和 vel, 点类的模式变量取值范围为点表, 即 T, f_T="点", 记这个表为 T_p. 线类的模式变量取值范围为线表, 即 T, f_T="线", 记这个表为 T_l.

定义 11(模式变量表). 模式变量表是一个有序集合, 包含所有在查询任务中模式部分出现的变量, 记为 VS. VS 中第 i 个元素记为 ve_i, VS=(ve_1,ve_2,ve_3,…).

在操作用例中, 模式变量表可以为(A,B,C,a,b), 模式变量表本身定义为有序集合, 但是其顺序与变量出现关系无关.

定义 12(模式变量取值空间). 对于模式变量表 VS, ve_i 的取值范围为 T_i, 如果 VS 有 n 个元素, 则模式变量取值空间则为笛卡尔积 $T_1\times T_2\times T_3\times\ldots\times T_n$, 记为 T^*.

有了模式变量取值空间, 可以进行第 1 步操作, 根据查询任务中的第 3 部分"条件", 对于 n 个元素的模式变量表, 可以筛选出符合"条件"的 n 元有序元组(t_1,t_2,t_3,…,t_n)的集合, 这个集合是模式变量取值空间的子集, 称之为候选空间.

定义 13(取属性操作). 对于表述"A[B]", 称之为取属性操作, 其数学含义为:

变量 A 的取值为元组 t 时, 进行分类讨论: 如果$\exists e\in t$, 使得 K_e=B, 则 A[B]=V_e; 如果$\forall e\in t$, $K_e\in B$, 则 A[B]=not_exist, not_exist 为特殊值. 因此, A[B]$\in val\cup\{not\_exist\}$.

取属性操作可以迭代, 例如: 当 A[B]$\in address$ 时, A[B]视为该地址的元组, 此时 A[B][C]的表述也是合法的, 以此类推.

定义 14(子条件). 对于每个被逗号分隔的条件称为子条件, 子条件是逻辑连词符号的传统数学条件公式的表述, 因此对于任何模式变量表的取值, 可以返回布尔变量真或假, 即 c:$T^*\rightarrow$(TRUE,FALSE). 有一个例外情况需要额外讨论. 特别定义表述"$A.not\_has$(B)", 意为"A 不含属性 B", 当 A[B]=not_exist 时, 该表述返回布尔变量真. 当 A[B]=not_exist 时, 对包含变量 A[B]的条件判断均返回布尔变量假.

具体而言, 当 A[B]=not_exist 时, "$A.not\_has$(B)"为真, "A[B]>10"为假, "A[B]≠'M'"为假, 但是"(35>22)∨(A[B]>10)"为真.

定义 15(条件序列). 条件序列是查询任务中的"条件"产生的所有子条件的且序列, 因此也能返回布尔变量真或假, 记为 cl, 若子条件集合为(c_1,c_2,c_3,…,c_n), cl=$c_1\wedge c_2\wedge c_3\wedge\ldots\wedge c_n$.

定义 16(合法取值). 对于变量模式表 VS=(ve_1,ve_2,ve_3,…,ve_n), 现有 n 元元组 t^*, t^*=(t_1,t_2,t_3,…,t_n), $t^*\in T^*$, 如果对$\forall i\in[1,n]$, ve_i 的取值为 t_i, 则称 VS 取值为 t^*, 当 VS 取值为 t^*时, 条件序列返回值为真, 则称 t^*为合法取值.

定义 17(候选空间). 对于模式变量取值空间 T^*, 有候选空间 $P^*\subset T^*$, P^*为所有合法取值的集合.

候选空间 T^*中所包含的多元元组, 其实就是查询任务中符合"条件"的模式变量表的取值, 接下来需要考虑匹配"模式"的相关问题.

定义 18(子模式). 对于查询任务的"模式"部分, 可以视为由逗号分隔的每个子模式, 子模式是由类似"(A)-[a]→(B)"的符号构成的, 类似于子条件, 其对于模式变量表的取值也可以返回布尔变量真或假, 即 $T^*\rightarrow$(TRUE,FALSE), 其依据在于该取值是否匹配该子模式.

具体而言, 当面对"(A)-[a]→(B)"这个子模式时, a 的取值如果是线基类元组 t_a, 则判断 t_a 的起点是否为 A 的取值 t_A 的地址, 判断 t_a 的终点是否为 B 的取值 t_B 的地址. 若二者均为真, 则此子模式返回值为真.

子模式具有特殊表达方式, 例如用例中的"(A)←[a]-(C)-[b]→(B)", 其等价于"(C)-[a]→(A)"且"(C)-[b]→(B)"且"$a\neq b$"且"A, B, C 三者之间互不相等". 这样的特殊表达可以极大地提升查询语句的表达能力.

例如: 现提出一个朋友关系图, 其中, 朋友均为双向关系, 点为个人. 如果需要查找"小明的朋友集合", 设 A 取值为小明所在元组, 线取值均为朋友关系, 模式则是简单的"(A)-[a]→(B)"返回所有的 B; 此时, 如果需要查找"小明朋友的朋友", 模式则是"(A)-[a]→(B), (B)-[b]→(C)"返回所有的 C. 在上述的条件下, C 中的取值一定会有小明自己; 但如果需要查找"小明的二阶朋友", 模式则是"(A)←[a]-(B)-[b]→(C)"返回所有的 C, 这时

有额外的隐藏条件“$A\neq C$”, 因此小明不会在 C 的取值之中.

定义 19(模式序列). 模式序列是查询任务中的“模式”产生的所有子模式的且序列, 因此也能返回布尔变量真或假, 记为 ml, 若子模式集合为$(m_1,m_2,m_3,\ldots,m_n)$, $ml=m_1\wedge m_2\wedge m_3\wedge\ldots\wedge m_n$.

定义 20(合模取值). 对于变量模式表 $VS=(ve_1,ve_2,ve_3,\ldots,ve_n)$, 现有 n 元元组 t^*, $t^*=(t_1,t_2,t_3,\ldots,t_n)$, $t^*\in P^*$, 如果对$\forall i\in[1,n]$, ve_i 的取值为 t_i, 则称 VS 取值为 t^*, 当 VS 取值为 t^*时, 模式序列返回值为真, 则称 t^*为合模取值.

定义 21(匹配空间). 对于候选空间 P^*, 有匹配空间 $M^*\subset P^*$, M^*为所有合模取值的集合.

匹配空间 M^*中所包含的多元元组, 其实就是查询任务中符合“模式”和“条件”的模式变量表的取值, 接下来要做的, 就是将这些元组按查询任务中的“对象”进行返回即可.

定义 22(返回变量表). 基于查询任务的“对象”, 找到需要返回的模式变量, 有序集合为返回变量表, 返回变量表 RS 是模式变量表 VS 的子集, $RS\subset VS$, RS 中第 i 个元素记为 re_i, $RS=(re_1,re_2,re_3,\ldots)$.

假设返回变量表 RS 有 n 个元素, 模式变量表有 m 个元素, 可以获得一个大小为 n 的投影表 pro_tab, $pro\_tab[i]=k$, 当且仅当 $re_i=ve_k$.

对于匹配空间 P^*中的每一个多元元组 m^*, 均可以获得其在返回变量表中的投影 r^*, 记 r^*的第 i 个元素为 r_i, m^*中的第 i 个元素为 m_i, $r_i=m_{pro\_tab[i]}$.

定义 23(返回空间). 对于匹配空间 M^*, 其包含的每一个多元元组 m^*, 获得其投影 r^*, r^*的集合就是返回空间 R^*.

3.2 面向云边端协同的多模态数据查询操作

给出多模态数据查询操作的基础定义之后, 接下来介绍云边端协同架构中涉及的多模态数据典型查询. 云边端协同架构涉及查询复杂多样, 其中包括即席查询(ad-hoc queries)[9]与持续查询(continuous queries)[10]. 下面给出这两种查询的具体定义.

定义 24(即席查询). 在云边端场景下, 端侧数据库通常是传统关系型数据库的应用, 主要是在端侧对数据库中的数据进行 DML 操作[11]; 而边侧和云侧更类似于数据仓库, 支持对大量的多模态数据进行汇总. 在端侧可以进行查询与调整数据, 而在边侧和云侧更多是可以来做报表与统计预测[12]. 即席查询需求一般在涉及边侧与云侧场景下产生[13,14].

定义 25(持续查询). 持续查询是云边端数据库时序引擎内部自动定期执行的时序查询, 在时序数据应用场景下, 对于依照时间推进顺序写入的实时数据, 用户有时会希望每隔一段固定时间, 就能够按照一定的查询条件对该时间范围内的时序数据做一次计算(例如对该时间范围内的数据进行聚合计算), 并将计算结果保存下来[15].

这两种形式的云边端协同多模态查询的基本数据操作体系均包括选择、投影、笛卡尔积、并、交、差、分组、聚合以及结果提取操作. 即: 在进行即席查询与连续查询的过程中, 用户可以自行调用以上查询作为原子操作.

云边端各侧、边-边协同、端-端协同、云-边协同、边-端协同、云-边-端协同场景均会涉及多模态数据的选择、投影、笛卡尔积操作.

定义 26(选择). 表 T 上的数据选择操作定义为 $Sel_c(T)=(T',T_R)$, 其中, $T'=\{C_t,D(C_t,c)|t\in C_R\}$.

计算 V_t对约束 c 的满足程度, 计算方法如下.

- 若 c 形式为 $C_t\neq C_c$ 的原子约束(C_t,c_c 的定义域为 dom_C), 则对于$\forall c\in dom\cap c\neq c_c$, 都有 $D(C_t,c)=1$.
- 若 c 形式为 $C_t>C_c$ 的原子约束(C_t,c_c 的定义域为 dom_C), 则对于$\forall c\in dom\cap c>c_c$, 都有 $D(C_t,c)=1$.
- 若 c 形式为 $C_t<C_c$ 的原子约束(C_t,c_c 的定义域为 dom_C), 则对于$\forall c\in dom\cap c<c_c$, 都有 $D(C_t,c)=0$.
- 对于复合约束 $c=c_1*c_2$, $D(C_t,c)=\begin{cases}D(C_t,c_1)\times D(C_t,c_2), & *=\cap\\ 1-(1-D(C_t,c_1))\times(1-D(C_t,c_2)), & *=\cup\end{cases}$.
- 对于复合约束 $c=\neg c'$, $D(C_t,c)=1-D(C_t,c')$.

云边端协同各场景下选择操作的定义如表 1 所示.

表 1 云边端协同各场景下选择操作定义

架构	应用场景举例	选择定义
云	在工厂数据中选择电压大于 30 的车间数据	$Sel_c(T_{Cloud})=(T'_{Cloud},S_T)$
边	在车间数据中选择压力大于 60 的设备数据	$Sel_c(T_{Edge})=(T'_{Edge},S_T)$
端	在设备数据中选择温度大于 20 的数据	$Sel_c(T_{Device})=(T'_{Device},S_T)$
端-端协同	在多个设备之间联合选择温度大于 20 的数据	$Sel_c(Union(T_{Device},T_{Device}))=(C',S_T)$
云-边协同	在 A 和 B 工厂内的车间数据中联合选择温度大于 20 的数据	$Sel_c(Union(T_{Cloud},T_{Edge}))=(C',S_T)$
边-端协同	在 C 车间和 D 车间的设备之间联合选择温度大于 20 的数据	$Sel_c(Union(T_{Edge},T_{Device}))=(C',S_T)$
边-边协同	在多个车间之间联合选择温度大于 20 的数据	$Sel_c(Union(T_{Edge},T_{Edge}))=(C',S_T)$
云-边-端协同	在多个工厂、车间和设备数据中联合选择温度大于 20 的数据	$Sel_c(Union(T_{Cloud},T_{Edge},T_{Device}))=(C',S_T)$

定义 27(投影). 表 T 上的投影操作定义为: $\Pi_S T=(T',S)$, 其中, $T'=\{match(t,S)|t\in C_T\}$.

函数 $match(t,S)$将 t 转化成模式为 S 的数据, 如果 S 是 S_T 的子集, 则结果就是 t 在 S 上的投影; 否则, 需要模式转换.

$match(t,S)$定义为元组 t'对于每个属性 $s_a\in S$, 其属性:

$$t'[s_a]=t\left[\underset{t_a\in S_t}{arc\ equal}(t_a,s_a)\right],$$

其中, 函数 $\underset{t_a\in S_t}{arc\ equal}(t_a,s_a)$ 表示模式 S_t 中与 s_a 名字相同的属性.

云边端协同各场景下投影操作的定义如表 2 所示.

表 2 云边端协同各场景下投影操作定义

架构	应用场景	投影定义
云	在工厂数据表上投影模式 S	$\Pi_S(T_{Cloud})=(T'_{Cloud},S)$
边	在车间数据表上投影模式 S	$\Pi_S(T_{Edge})=(T'_{Edge},S)$
端	在设备数据表上投影模式 S	$\Pi_S(T_{Device})=(T'_{Device},S)$
端-端协同	在多个设备数据表上投影模式 S	$\Pi_S(Union(T_{Device},T_{Device}))=(C',S)$
云-边协同	在 A 工厂和 B 工厂内的车间数据表上投影模式 S	$\Pi_S(Union(T_{Cloud},T_{Edge}))=(C',S)$
边-端协同	在 C 车间和 D 车间的设备数据表上投影模式 S	$\Pi_S(Union(T_{Edge},T_{Device}))=(C',S)$
边-边协同	在多个车间数据表上投影模式 S	$\Pi_S(Union(T_{Edge},T_{Edge}))=(C',S)$
云-边-端协同	在多个工厂、车间和设备数据表上投影模式 S	$\Pi_S(Union(T_{Cloud},T_{Edge},T_{Device}))=(C',S)$

定义 28(笛卡尔积). 表 T 上的笛卡尔积操作定义为

$$cartesian(T_1,T_2)=(T',(S_{T_1},S_{T_2})),$$

其中, $T'=\{(C_{t_1},C_{t_2})\,|\,t_1\in C_{T_1},t_2\in C_{T_2}\}$. 与关系上的连接操作类似, 表 T 和 S 以 c 为约束的连接操作可以表示为 $Join(T_1,T_2,c)=Sel_c(T',(S_{T_1},S_{T_2}))$, 其中, $T'=\{(C,D(C,c))|C\in Cartesian(T_1,T_2)\}$

云边端协同各场景下笛卡尔积操作的定义如表 3 所示.

表 3 云边端协同各场景下笛卡尔积操作定义

架构	应用场景	笛卡尔积定义
端-端协同	在多个设备数据之间联合查询	$Join(T_{Device_1},T_{Device_2},c)=Sel_c(T',(S_{T_{Device_1}},S_{T_{Device_2}}))$
边-端协同	在 C 车间和 D 车间的设备数据之间联合查询	$Join(T_{Edge},T_{Device},c)=Sel_c(T',(S_{Edge},S_{T_{Device}}))$
边-边协同	在多个车间数据之间联合查询	$Join(T_{Edge_1},T_{Edge_2},c)=Sel_c(T',(S_{T_{Edge1}},S_{T_{Edge2}}))$
云-边协同	在 A 工厂和 B 工厂内的车间数据之间联合查询	$Join(T_{Cloud},T_{Edge},c)=Sel_c(T',(S_{T_{Cloud}},S_{T_{Edge}}))$
云-边-端协同	在多个工厂、车间和设备数据之间联合查询	$Join(T_{Cloud},T_{Edge},T_{Device},c)=Sel_c(T',(S_{T_{Cloud}},S_{T_{Edge}},S_{T_{Device}}))$

集合操作要求参与操作的集合模式相同. 在云边端各侧、边-边协同、端-端协同、云-边协同、边-端协同、云-边-端协同场景进行集合操作时, 如模式不相同, 要先进行模式转换. 集合操作并、交和差的定义见定义 29–定义 31.

定义 29(并). 表 T_1 和 T_2 的并定义为

$$Union(T_1,T_2)=(C_{T_1}\cup \Pi_{S_{T_1}}(C_{T_2}),S_{T_1}).$$

定义 30(交). 表 T_1 和 T_2 的交定义为

$$Intersection(T_1,T_2)=(C_{T_1}\cap \Pi_{S_{T_1}}(C_{T_2}),S_{T_1}).$$

定义 31(差). 表 T_1 和 T_2 的差定义为

$$Except(T_1,T_2)=(C_{T_1}-\Pi_{S_{T_1}}(C_{T_2}),S_{T_1}).$$

云边端各侧、边-边协同、端-端协同、云-边协同、边-端协同、云-边-端协同场景均会涉及分析操作.

定义 32(分组). (γ,A)分组: 二维张量 R 的(γ,A)分组 $S_{(\gamma,A)}$是一维张量集合, 其中, A 代表分组所依据的属性集合, γ代表在相应属性集合上的选择规则集合.

$$G_{(\gamma,A)}=\{t|t\in R\cap\forall t'\in G_{(\gamma,A)},t'\in Sel_{\gamma}(R)\}.$$

下面以端-端协同举例说明分组操作.

由于端侧设备计算能力的限制, 在数据上的分组不可以简单地套用现有算法, 而要在分组设计的过程中充分地考虑设备可用计算资源的有限性. 端侧分组的实际操作大致有思路如下, 即: 根据各个二维张量中每条数据在属性集合 A 上的取值, 在端-端协同选择操作获得的分组的基础上再进行相应的处理, 得到进行粗糙的分组. 其与端-端选择的具体差异在于: 该操作最后的分组结果会包括原有的所有数据样本, 而选择操作将只会得到所挑选出的部分.

定义 33(聚合). 聚合的实现依赖于聚集函数(aggregate function). 聚集函数可以用 *fun* 表示, 使用方法是输入值的一个集合, 将单一值作为结果返回. 例如, 计算平均值(avg)或排序(sort).

聚集运算(aggregation operation) G 的定义如下.

关系集合 R 以聚集函数 *fun* 为聚集函数, 在属性集合 A 上γ聚集运算定义为

$$G(R,A,\gamma,fun)=(R',fun(A)),$$

其中, R'是根据分组运算 $S_{(\gamma,A)}$计算得到的结果.

该聚集运算 G 的基本思想是: 在基本聚集函数的基础上, 对云边端数据处理做了一层封装. 即: 在聚集运算 G 的过程中, 其将会调用基本的聚集函数 *fun*, 并根据云边端数据的特点做额外的附加处理.

云边各侧、边-边协同、云-边协同、云-边-端协同场景均会涉及结果提取操作.

与单一的关系数据库不同, 云边端关系数据库上的查询结果有具体的场景要求, 而由操作产生的结果不一定能够满足查询中的具体场景要求. 在工业生产中经常设定合格率大于阈值ε的条件, 并依据该条件进行结果提取. 因此, 需要有比较普遍的结果提取函数对查询操作生成的结果进行提取, 得到满足要求的结果.

下面定义 3 种结果提取操作.

定义 34(*A* 提取). 对于关系 R, A 提取定义为

$$Extraction_A(R)=(R',S_R),$$

其中, A 代表从用户端输入的某种属性集合, 其取值可以包含产品合格率等; S_R 代表关系的模式; 而 R'代表 A 提取的目标集合, 且该集合的 A 集合中的属性满足一定的规则集合γ. 即有如下定义.

$$R'=\{t|t\in R\cap(\forall a_i\in A,\ t.a_i \text{ follows } \gamma)\}.$$

例如: 如果想要提取产品合格率高于一定阈值ε_0的产品, 则 A 与γ有如下定义.

$$A=\{\varepsilon\},$$

$$\gamma=\{\varepsilon>\varepsilon_0\}.$$

此时, R'的定义如下:

$$R'=\{t|t\in R\cap t.\varepsilon>\varepsilon_0)\}.$$

定义 35(*TopK* 提取). 对于关系 R, *TopK* 提取定义为

$$Extraction_{TopK}(R)=(R_E,S_R),$$

其中, R_E 满足下列 3 个条件: (1) $R_E\subseteq R$; (2) $|R_E|=k$; (3) $\forall t\subseteq R_E$, $\forall t'\subseteq R_R-R_E$, $\varepsilon_t\geqslant\varepsilon_{t'}$.

定义 36(*k* 重要提取). 对于表 T, k 重要提取定义为

$$Extraction_{significant\text{-}k}(G)=(T_E,S_T),$$

其中, R_E 满足下列 3 个条件: (1) $R_E \subseteq R$; (2) $|R_E|=k$; (3) $\forall R'_E \subseteq R$, 满足 $|R'_E|=k$, 满足:

$$\sum \forall t_1,t_2 \in R_E dist(v_{t_1},v_{t_2}) \leqslant \sum \forall t_1,t_2 \in R'_E dist(v_{t_1},v_{t_2}).$$

一般认为, 数据时序信息越新, 与正常数据相差越大的数据条目越具有重要价值. 因此, 将 k 重要提取中 *dist* 定义如下.

$$dist(v_1,v_2)=w_1\times(t_2-t_1)+w_2\times res(v_2,v_1),$$

其中, w_1 是时序信息占比权重, w_2 是 $res(v_2,v_1)$占比权重.

3.3 面向云边端协同的多模态数据变化操作

面向云边端协同的多模态数据变化操作主要包括插入操作、删除操作和修改操作.

删除操作是从元组的角度出发进行命名的, 对于某个元组中元素的增删改, 均视为对于元组的修改. 因此, 数据删除定义为对于元组的删除.

定义 37(删除操作). 基于查询操作获得的返回空间 R^*与返回变量表 *RS*, 选取变量 *A*, 其中, $A \in RS$, *A* 在 R^* 中的所有取值是一个元组集合, 对于这个集合中所有元组, 在数据库和表中的删除.

定义 38(添加操作). 基于查询操作获得的返回空间 R^*与返回变量表 *RS*, 选取变量 *A*, 其中, $A \in RS$, *A* 在 R^* 中的所有取值是一个元组集合, 记为 *T*. 可以对 *T* 中每个元组进行类似于 for 循环的程序操作, *T* 中元组的地址可以作为新元组中的某些元素的“值”, 即提供外键, 从而添加新元组.

定义 39(修改操作). 基于查询操作获得的返回空间 R^*与返回变量表 *RS*, 选取变量 *A*, 其中, $A \in RS$, *A* 在 R^* 中的所有取值是一个元组集合, 记为 *T*.

对于元素的增删改, 给出对应 K_e 基本可以实现. 删除时, 对于给定的 $K_{ed}=key\_to\_delete$, 删除掉该元组 $K_e=key\_to\_delete$ 的元素即可; 增加时, 对于给定的 $K_{ea}=key\_to\_add$ 和 $V_{ea}=value\_to\_add$, 增加一个元素 $e=(K_{ea}, V_{ea})$即可; 修改时, 视为对元素的删除操作和增加操作前后顺序叠加执行即可.

4 多模态数据模型完整性约束

本文对多模态数据模型的完整性约束进行了研究, 主要包括主键约束、外键约束、范式与事务及其 ACID.

4.1 主键约束

对于数据库 *D*, 在其数学定义的基础上, 完整性约束提供一些新的规则, 会对表和元组产生新的限制和性质, 主键约束体现的是完整性约束中实体完整性. 在本文提出的多模态数据模型中, 主键为组合主键, 必须包含但不能仅包含“基类”和“类型”两个元素, 一旦规定某个类型的主键, 主键元素不得缺失, 任何两个同一类的元组之间, 主键不得相同.

方便地, 称“基类”和“类型”两个元素完全相同的元组为同类. 对于一个类型的数据, 可以约定主键, 对于同类元组的主键约束其实可以将其视为关系型数据库中的一个表, 主键的额外部分即是关系型数据库中的主键, 同一类元组, 其主键元素有着相同的“键”.

定义 40(元组类型). “基类”和“类型”两个元素完全相同的元组为同类, 记为 *T*. 同一类元组在数学上可以组成表, 但是表中往往有多个类型的元组.

定义 41(主键). 主键是基于一类元组的概念, 记为 MK_T, 是一个 K_e 的有序集合, 其中, 第 1 个元素为“基类”, 第 2 个元素为“类型”, $size(MK_T)\geqslant 3$.

定义 42(主键约束). 对数据库 *D* 中所有元组 *t* 作如下约束: 基于其“基类”和“类型”决定其类型 *T*, 找到对应主键 MK_T, 利用取属性操作获得其主键对应取值 MK_{T_t}, MK_{T_t} 中不允许出现 *not_exist* 值, *D* 中$\forall t_1 \neq t_2$, 有:

$$MK_{T_{t_1}} \neq MK_{T_{t_2}}.$$

对于点基类的元组, 因为在多模态数据库的要求中, 其需要履行属性结构的要求, 因此建议对每个类型进行主键的约定. 线基类的元组则视情况而定, 若其也有明显的属性结构, 也推荐进行主键的约定. 而对于属

性基类元组、时序基类的元组, 其发挥作用往往在于被解析地址的时候, 一般不约定主键. 当然, 如果需要的话, 依然可以支持主键约束.

4.2 外键约束

外键约束体现了数据模型完整性约束的参照完整性, 其强调数据与数据间的参照依赖关系. 在本课题的数据模型中, 这一特性集中体现在对于地址索引的使用上. 因此, 首先对大部分地址提出一个最基本的要求:

定义 43(外键约束). 地址所在物理空间首先要存有元组, 而不是一个“无效地址”或者“野指针”. 此外, *NULL* 是地址的合法取值, 但在某些情况下不符合一些基类的要求.

对于数据库 *D*, 基于其各自的基类, 划分了不一样的表, 每个表中有着相同基类的元组. 为了让这一数学模型有意义, 自然要对不同的基类作不同的外键约束.

- 点基类: 点基类元组必须包含一个以“连线”为“键”的元素, 该元素必须指向线基类元组或者为 *NULL*.
- 线基类: 线基类元组必须分别包含以“起点”“终点”为“键”的两个元素, 这两个元素必须指向点基类元组, 且不能为 *NULL*. 除此以外, 必须分别包含以“起点前一线”“起点后一线”“终点前一线”“终点后一线”为“键”的 4 个元素, 这 4 个元素必须指向线基类元组或者为 *NULL*.

为了保证图数据库的处理效率, 在物理存储这些元组时, 建议在元组的固定位置存放以上元素, 并做出规定, 以便程序上按位进行存取, 可以极大地提升图数据库的处理速度.

4.3 范　式

基于关系型数据库具有的 5 种范式, 本文也对多模态数据模型做出了一些范式的可能性探索.

- 基类决定类型: “基类”和“类型”两个元素一直起着至关重要的作用, 尤其是在主键约束部分. 实际上, “类型”部分是本模型留给用户自由度极高的自命名部分, 用户可以根据类型规定元组的主键, 甚至在一定程度上定义了某些元组的“模式”. 但如果同一个“类型”名称同时出现在不同的基类当中, 这往往意味着将现实模型抽象为数据模型的过程中的某些环节出现了问题, 无论是抽象的过程或者是命名的过程.
- 地址指向的基类: 在外键约束部分, 对于点和线的基类中必要的地址元素进行了规定. 除此以外, 元组还可以拥有许多非必要的元素, 这些元素的取值可以是地址, 这些地址往往指向属性基类元组、时序基类元组、编码基类元组. 一个点元组的某些元素, 可以指向其他点元组, 这看起来似乎非常灵活, 但实际上, 如果一个点元组通过地址指向其他点元组, 这往往意味着这二者之间具有某种关系. 而在本文提出的数据模型中, 这种关系已经有最合适的表达方式—线基类. 因此, 当一个点元组通过非必要的元素指向另一个元组的时候, 这种信息并不能被图结构所解读. 当这种行为的数量到达一定规模时, 意味着整个数据库变成了一种几乎难以维护的状态.

为了避免上文所述现象, 则需要定下新的范式: 对非必要元素取值为地址时, 地址目标种类进行限制. 它们可以是属性基类元组、时序基类元组、编码基类元组, 但不能是点基类、线基类元组.

4.4 事务及其ACID

事务指的是一个不可分割的逻辑工作单元, 允许上层业务将多个操作作为一个整体执行. 将面向云边端协同的多模态数据操作进行事务化, 则依然必须遵从 4 个属性: 原子性(atomicity)、一致性(consistency)、隔离性(isolation)、持久性(durability).

由于云边端协同场景下具有更加复杂的操作, 并且各侧设备和资源存在异构性, 所以云边端协同应用中的事务特性与传统数据库事务相比, 存在的挑战主要包括以下几个方面.

(1) 原子性

原子性是指事务是一个不可分割的工作单位, 事务中的操作要么全部成功, 要么全部失败. 在云边端协同场景下, 特别是云-边-端协同操作时, 事务一般具有较长的生命周期, 传统数据库事务的原子性要求可能过于严格, 会造成时间、资源的大量浪费. 可以在锁方面进行优化.

(2) 一致性

一致性是指事务必须使数据库从一个一致性状态变换到另外一个一致性状态. 由于在云边端协同场景中存在大量并发操作, 所以不必要求系统时刻处于一致性状态. 但是要保证即使出现不一致情况, 状态仍然是可恢复的. 可以在执行事务前对于数据库 D 整体状态的保存, 使得系统在事务被中断或者执行失败的情况下具有恢复的能力.

(3) 隔离性

隔离性是多个用户并发访问数据库时, 数据库为每一个用户开启的事务, 不能被其他事务的操作数据所干扰, 多个并发事务之间要相互隔离. 但在云边端协同场景中, 协同并发执行的活动之间常常需要共享资源, 要求访问的资源状态在事务没有完成时就对外可见. 为了保证系统的并发度, 需要实现不过于严格隔离性, 这需要在读写锁方面进行优化.

(4) 持久性

持久性是指一个事务一旦被提交, 它对数据库中数据的改变就是永久性的. 与传统数据库类似, 对于云边端协同场景来说, 在事务执行的过程中, 一旦某个活动或者子过程成功执行完毕, 其结果也必须被持久化, 即便执行中出现过失效.

5 面向智能工厂的多模态数据模型应用示例

本文以云边端协同架构的智能工厂场景为例, 来说明云边端多模态数据模型与操作, 该场景如图 8 所示. 云边端协同架构的智能工厂中存在多种数据格式和数据类型的数据, 包括关系型数据、图数据、图像、视频、时序数据以及流数据等.

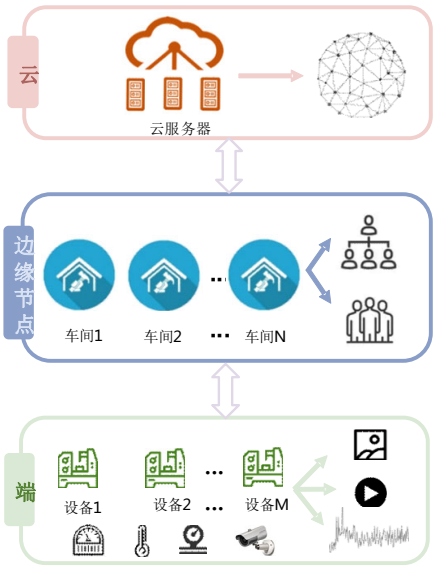


图 8 云-边-端协同架构智能工厂示意图

云边端三层数据用例分别见表 4–表 11, 数据共有 6 种“基类”, “基类”的集合为(点、线、属性、高维时序、时序、编码), 每个“基类”里面还有不同数据类型.

端侧主要采集性能数据, 涉及生产安全监控和装配质量检测. 表 4 存储的是端侧设备采集的时序数据, 即 102 号车间 841 号设备的电压、温度以及压力数据; 表 5 存储的是端侧采集的监控数据, 即 102 号车间 841 号设备的监控图片数据.

表 4 端侧设备 102-841 传感器数据

地址	基类	类型	车间编号	设备编号	2019-04-18 09:00:00	2019-04-18 10:00:00	2019-04-18 11:00:00	2019-04-18 12:00:00
0x01	时序	电压	102	841	383	375	370	379
0x02	时序	温度	102	841	18	23	19	22
0x03	时序	压力	102	841	45	78	67	34

表 5 端侧设备 102-841 监控数据

地址	基类	类型	车间编号	设备编号	编码方式	编码	大小	分辨率
0x04	编码	监控图片	102	841	JPG		23KB	300*400
0x05	编码	监控图片	102	841	JPG	…	27KB	300*400

表 6 边侧 102 车间传感器数据

地址	基类	类型	车间编号	设备编号	电压	温度	压力
0x06	高维时序	传感器	102	841	0x01	0x02	0x03
0x07	高维时序	传感器	102	842	0x04	0x05	0x06

表 7 边侧 102 车间员工数据

地址	基类	类型	工号	姓名	连线	年龄	性别	部门	车间编号
0x08	点	员工	101	张三	0x19	45	男	运输部	102
0x09	点	员工	102	李四	0x20	37	男	运输部	102
0x10	点	员工	103	王七	0x21	36	女	仓管部	102

表 8 边侧车间 102 物料入库数据

地址	基类	类型	名称	连线 1	连线 2	入库单号	员工	供应商	监控图片
0x11	点	物料	曲轴	0x19	0x22	20190418	0x08	0x14	0x04
0x12	点	物料	连杆	0x20	0x23	20190419	0x09	0x15	0x05
0x13	点	物料	曲轴	0x21	0x24	20190420	0x10	0x14	...

表 9 物料供应商数据

地址	基类	类型	名称	分类	信誉等级	注册时间
0x14	属性	供应商	A01	生产商	A	2002-01-02
0x15	属性	供应商	A02	加工商	B	2003-05-03

表 10 云侧传感器数据

地址	基类	类型	车间编号	设备编号	电压	温度	压力
0x16	高维时序	传感器	102	841	0x01	0x02	0x03
0x17	高维时序	传感器	102	842	...	...	...
0x18	高维时序	传感器	241	134	...	...	...

表 11 物料入库分析数据

地址	基类	类型	编号	起点	终点	起点前一线	起点后一线	终点前一线	终点后一线
0x19	线	物料入库	1	0x08	0x11	NULL	NULL	0x22	NULL
0x20	线	物料入库	2	0x09	0x12	NULL	NULL	0x23	NULL
0x21	线	物料入库	3	0x10	0x13	NULL	NULL	0x24	NULL
0x22	线	物料入库	4	0x14	0x11	NULL	NULL	0x19	NULL
0x23	线	物料入库	5	0x15	0x12	NULL	0x24	0x20	NULL
0x24	线	物料入库	6	0x14	0x13	0x23	NULL	0x21	NULL

边侧主要存储某个车间的数据, 一部分由端侧上传, 用于生产进度查询、制品质量追溯等, 一部分为车间员工信息表、物料入库等信息表. 表 6 存储的是边侧设备数据, 即 102 号车间的设备数据; 表 7 存储的是边侧人员信息, 即 102 号车间的员工数据. 表 8 存储的是物料入库数据, 即 102 号车间的物料入库数据; 表 9 存储的是物料供应商信息.

云侧主要负责存储由边端侧上传的数据, 并进行相关工艺参数优化、物料入库分析等. 表 10 存储的是云侧传感器数据, 即该工厂所有设备的全部数据. 表 11 存储的是物料入库分析数据.

6 模型的实现与实验验证

为了验证本文提出的多模态数据建模技术的可行性, 对多模态数据模型的存储性能进行了验证. 接下来, 首先给出实验环境设置以及多模态数据集的基本内容; 然后, 从存储时间、存储空间和查询时间这 3 个方面对本文提出的数据模型与 PostgreSQL 数据库进行了对比实验, 并对实验结果进行分析.

6.1 实验环境设置及数据集

实验基于 Python 3.10 和 10.23 版本的 PostgreSQL 开发, 在台式计算机进行, 硬件配置为 CPU Intel(R) Core (TM)10875H 1.8 GHz; 内存 16.00 GB.

由于现在没有同时包含图数据、关系数据等多模态数据的标准公开数据集, 本文以社交网络为场景构建了多模态数据集, 主要包括图数据与关系数据. 图数据中的结点数据由若干个员工 id 组成, 边数据代表用户之间存在 related 关系, 关系数据由员工 Gender, Age, Organization 等 10 个属性的个人信息组成. 在云边端协同场景下, 员工的基本信息等关系数据通常存储在边侧, 描述员工之间关联的图数据通常可以存储在云侧, 用于员工架构调整与任务分配等. 由此可见, 使用该数据集评价本文所提出的面向云边端协同的多模态数据模型比较合理. 数据集的具体信息见表 12.

表 12 实验数据集

Dataset	Number of nodes	Number of edges	Number of relationships	Size of data (KB)
Dataset1	7 604	24 186	4 356	443
Dataset2	9 994	43 703	5 630	803
Dataset3	13 697	61 957	6 389	1 141
Dataset4	18 017	73 338	7 522	1 382

6.2 实验结果分析

基于 PostgreSQL 数据库对多模态数据模型进行初步实现, 实验将上述 4 个数据集分别以多模态数据模型方式和传统关系模型方式存储到 PostgreSQL 数据库中, 然后从存储时间、存储空间和查询时间这 3 个方面验证方法的可行性. 为了保证实验的准确性, 本文实验数据均取 5 次实验结果的平均值.

- 存储时间

将 4 个数据集按上述两种方案存储, 所花费的时间如图 9 所示.

从图 9 可以看出, 两种数据模型的存储时间均随数据规模的增大而线性增长. 然而, 多模态数据模型存储数据的时间比关系数据模型长, 这是因为关系数据模型的存储方式是将图数据中的点和线分为两个表进行存储; 而多模态数据模型存储数据时, 需要对多模态数据进行分析, 分析过程造成一定的时间消耗.

- 存储空间

将 4 个数据集按上述两种方案存储, 存储空间占用情况如图 10 所示.

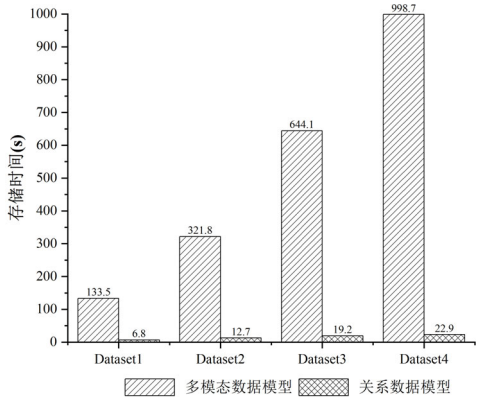


图 9 存储时间

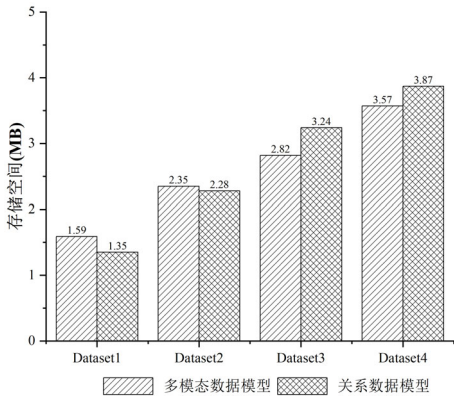


图 10 存储空间

从图 10 可以看出, 两种数据模型的内存占用大小都随着数据集规模的增大呈线性增长. 使用多模态数据模型存储 Dataset1 和 Dataset2 时的存储空间占用比关系数据模型分别多 17%和 3%, 而存储 Dataset3 和 Dataset4 时, 多模态数据模型的存储空间占用比关系数据模型低 12%和 7%. 说明随着多模态数据规模的增大, 使用多模态数据模型存储数据的内存占用要低于关系数据模型.

- 查询时间

设计 3 种多模态数据查询负载, Query1、Query2 和 Query3 中涉及图数据的边数满足线性增长. 3 种查询分别在以多模态数据模型和关系数据模型存储的 4 个数据集上执行, 查询执行时间如图 11–图 13 所示.

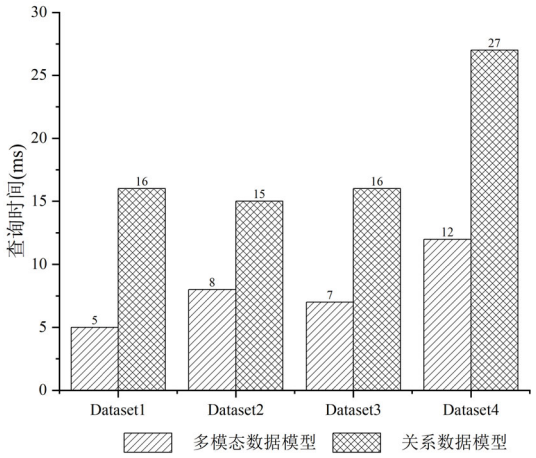


图 11 Query1 执行时间

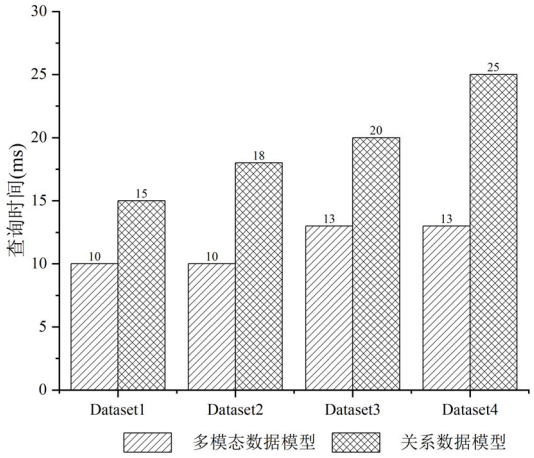


图 12 Query2 执行时间

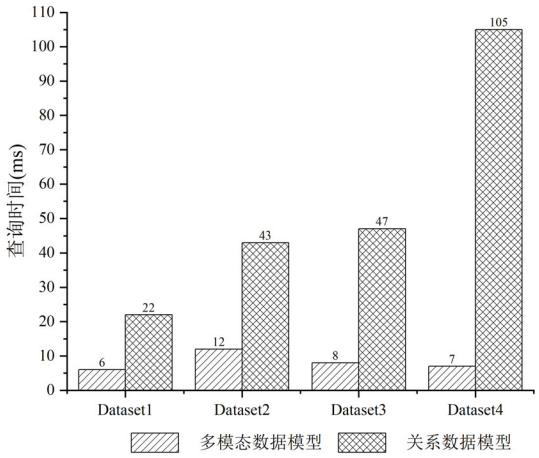


图 13 Query3 执行时间

整体来看, 一方面, 多模态数据模型的查询时间均短于关系数据模型; 另一方面, 在查询负载复杂程度不一致、数据规模逐渐变大的情况下, 多模态数据模型的查询时间依然较为稳定. 而当查询负载复杂、数据规模变大时, 关系数据模型的查询会耗费更久的时间.

例如: 在关系数据模型存储的 Dataset4 上执行 Query3, 查询执行时间是 105 ms, 而多模态数据模型只需要 7 ms, 时间缩短了近 93%. 由此可见, 多模态数据模型的查询性能优于关系数据模型.

通过上述实验验证, 本文提出的多模态数据模型能够有效地表示多模态数据, 并且能加速多模态数据的查询.

7 总结与展望

本文提出了一种基于元组的面向云边端协同的多模态数据模型, 能够描述云边端单侧的多模态数据以及云边端三层多模态数据的协同依赖关系; 然后提出了面向云边端协同的多模态数据操作体系, 包括查询操作和插入、删除、修改操作, 满足云边端独立查询操作以及云边端协同查询操作; 并且对多模态数据模型的完整性约束进行了研究, 包括主键约束、外键约束、范式与事务及其 ACID; 最后, 以智能工厂为例, 介绍了多模态数据模型的示范应用. 并且, 从多模态数据存储时间、存储空间和查询时间这 3 个方面对所提出的数据模型存储方法进行了验证. 实验结果表明, 本文提出的建模技术比现有方法更有效且更具扩展性.

未来计划在多模态数据模型的基础上, 对面向云边端协同查询处理的查询优化开展进一步研究. 面向云边端协同查询处理的查询优化具有协作约束复杂、优化目标多样、数据高维高速、设备网络异构和节点稳定性低等挑战[8,16,17], 目前尚缺少能够应对云边端数据异构挑战的协同查询优化技术[18,19]. 因此, 面向云边端的协同查询处理和优化研究可以从面向云边端异构协同的多目标智能协同查询优化和高可靠查询优化等方面进行展开.

References:

[1] Hu Y. When the inflection point arrives, edge computing goes from 1.0 to 2.0. 2021 (in Chinese). https://www.ccidcom.com/yaowen/20201228/Un7zMy9mYNIWMTpzO180ffiw29vok.html

[2] Yu H, Liang ZT, Yan YC. Review on multi-source and multi-modal data fusion and integration. Information Studies: Theory & Application, 2020, 43(11): 169−178. (in Chinese with English abstract). [doi: 10.16353/j.cnki.1000-7490.2020.11.027]

[3] Sun YY, Jia ZT, Zhu HY. Survey of multimodal deep learning. Computer Engineering and Applications, 2020, 56(21): 1−10 (in Chinese with English abstract). [doi: 10.3778/j.issn.1002-8331.2002-0342]

[4] Date CJ. A Guide to the SQL Standard. 2nd ed., Addison-Wesley, 1989.

[5] Bray JPT, Sperberg-McQueen CM, Yergeau F. Extensible Markup Language (xml) 1.0 (Third Edition). W3C Recommendation, 2004. http://www.w3.org/TR/REC-xml/

[6] van Rest O, Hong SP, Kim JH, *et al*. PGQL: A property graph query language. In: Proc. of the 4th Int'l Workshop on Graph Data Management Experience and Systems (GRADES). 2016. 1−6. [doi: 10.1145/2960414.2960421]

[7] Gueni B, Abdessalem T, Cautis B, *et al*. Pruning nested XQuery queries. In: Proc. of the Conf. on Information and Knowledge Management (CIKM). 2008. 541−550.

[8] Wang X, Zou L, Wang CK, Peng P, Feng ZY. Research on knowledge graph data management: A survey. Ruan Jian Xue Bao/Journal of Software, 2019, 30(7): 2139−2174 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5841.htm [doi: 10.13328/j.cnki.jos.005841]

[9] Koolen M, Kamps J. The importance of anchor text for ad hoc search revisited. In: Proc. of the 33rd Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR 2010). New York: Association for Computing Machinery, 2010. 122−129. [doi: 10.1145/1835449.1835472]

[10] Chen FS, Zhang PF, Yu CC, *et al*. Path-based continuous spatial keyword queries. Complexity, 2022, 2022: Article ID 4091245. [doi: 10.1155/2022/4091245]

[11] Fujiwara Y, Nakatsuji M, Shiokawa H, *et al*. Efficient ad-hoc search for personalized PageRank. In: Proc. of the 2013 ACM SIGMOD Int'l Conf. on Management of Data (SIGMOD 2013). New York: Association for Computing Machinery, 2013. 445−456. [doi: 10.1145/2463676.2463717]

[12] Chen J, Liu YQ, Fang Y, *et al*. Axiomatically regularized pre-training for ad hoc search. In: Proc. of the 45th Int'l ACM SIGIR Conf. on Research and Development in Information Retrieval (SIGIR 2022). New York: Association for Computing Machinery, 2022. 1524−1534. [doi: 10.1145/3477495.3531943]

[13] Dai ZY, Callan J. Context-aware document term weighting for ad-hoc search. In: Proc. of the Web Conf. 2020 (WWW 2020). New York: Association for Computing Machinery, 2020. 1897−1907. [doi: 10.1145/3366423.3380258]

[14] Rencis E. On keyword-based ad-hoc querying of hospital data stored in semistar data ontologies. Procedia Computer Science, 2018, 138: 27−32. [doi: 10.1016/j.procs.2018.10.005]

[15] Wang X, Shi PR, Li JQ, *et al*. Privacy-preserving mechanisms of continuous location queries based on LBS: A comprehensive survey. In: Proc. of the 27th Int'l Conf. on Automation and Computing (ICAC). 2022. 1−6. [doi: 10.1109/ICAC55051.2022.9911102]

[16] Deligiannakis A, Kotidis Y, Roussopoulos N. Hierarchical in-network data aggregation with quality guarantees. In: Proc. of the Extending Database Technology (EDBT). 2004. 658−675.

[17] Madden S, Franklin MJ, Hellerstein JM, *et al*. TinyDB: An acquisitional query processing system for sensor networks. ACM Trans. on Database Systems, 2005, 30(1): 122−173. [doi: 10.1145/1061318.1061322]

[18] Silberstein A, Yang J. Many-to-many aggregation for sensor networks. In: Proc. of the IEEE 23rd Int'l Conf. on Data Engineering (ICDE). 2007. 986−995. [doi: 10.1109/ICDE. 2007.368957]

[19] Tang XY, Xu JL. Extending network lifetime for precision-constrained data aggregation in wireless sensor networks. In: Proc. of the IEEE 25th Int'l Conf. on Computer Communications (INFOCOM). 2006. 1−12. [doi: 10.1109/INFOCOM.2006.149]

附中文参考文献:

[1] 胡媛. 迎接拐点到来, 边缘计算从 1.0 到 2.0. 2021. https://www.ccidcom.com/yaowen/20201228/Un7zMy9mYNIWMTpzO180ffiw29vok.htm

[2] 余辉, 梁镇涛, 鄢宇晨. 多来源多模态数据融合与集成研究进展. 情报理论与实践, 2020, 43(11): 169−178.

[3] 孙影影, 贾振堂, 朱昊宇. 多模态深度学习综述. 计算机工程与应用, 2020, 56(21): 1−10. [doi: 10.3778/j.issn.1002-8331.2002-0342]

[8] 王鑫, 邹磊, 王朝坤, 彭鹏, 冯志勇. 知识图谱数据管理研究综述. 软件学报, 2019, 30(7): 2139−2174. http://www.jos.org.cn/1000-9825/5841.htm [doi:10.13328/j.cnki.jos.005841]

崔双双(1997－), 女, 博士生, CCF 学生会员, 主要研究领域为云边端协同数据库, 数据库智能存储策略.

王宏志(1978－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为数据库管理系统, 大数据分析与治理.

吴限(2000－), 男, 学士, 主要研究领域为复杂网络, 多模态数据库.

吴昊(2001－), 男, 本科生, CCF 学生会员, 主要研究领域为边缘计算.