文章编号:1000-5641(2014)05-0290-11

分布式环境中数据库模式设计实践

庞天泽, 张晨东, 高 明, 宫学庆
(华东师范大学 软件学院,上海 200062)

摘要: 近年来,数据规模呈爆炸式增长,使得传统集中式数据库难以满足业务需求.而分布式数据库可以将数据存储在多个节点上,具有更好的扩展性,从而可以支撑业务的不断增长.目前,许多企业已经开发出了成功的分布式数据库产品,例如 Google Spanner、淘宝的 OceanBase 等.传统数据库模式设计中,三大范式(1NF、2NF 和 3NF)及其扩展范式能够减少数据冗余和更新异常,并保证数据的完整性.然而,在分布式架构下,严格遵循范式的模式设计可能带来查询效率较低等问题,而使用反范式模式设计方法通常可以有效提高查询效率.OceanBase 是淘宝自主研发的分布式数据库,支持跨行跨表事务,并在 OLTP 中具有良好的性能,但是对于 OLAP 业务,其性能并不高.本文将以 OceanBase 为例,介绍如何利用反范式设计分布式数据库模式,以改善 OLAP 的查询性能,并通过在 OceanBase 上部署 TPC-H 基准评测验证了反范式模式设计的有效性和高效性.

关键词: 反范式; 分布式数据库; OceanBase; TPC-H
中图分类号: TP392 **文献标识码**: A **DOI**:10.3969/j.issn.1000-5641.2014.05.026

Implementation of database schema design in distributed environment

PANG Tian-ze, ZHANG Chen-dong, GAO Ming, GONG Xue-qing
(*Software Engineering Institute, East China Normal University, Shanghai* 200062, *China*)

Abstract: Recently, we have witnessed an exponential increase in the amount of data. It results in a problem that a centralized database is hard to scale-up to the massive business requirements. A distributed database (DDB) is an alternative that can be scalable to the large scale applications by distributing the data to multi-node server. Now, many enterprises have successfully implemented some distributed databases, such as Google Spanner and TaoBao OceanBase. In the theory of the designation of traditional database, different normal forms reduce the operational exception and data redundancy, and also ensure the data integrity. However, a schema design strictly following the normal forms leads to an inefficiently distributed database system because of the large amount of distributed relational operations. Fortunately, denormalization can significantly improve the query efficiency by reducing the number of relations and the amount of the distributed relational operations. OceanBase, a distributed database, is implemented by TaoBao and has high performance for OLTP, rather than OLAP. In this paper, we introduce how to utilize de-

收稿日期:2014-07
基金项目:国家 973 课题(2010CB731402)
第一作者:庞天泽,男,硕士生,研究方向为分布式数据库. E-mail:pangtz@ecnu.edu.com.
通信作者:宫学庆,男,教授,博士生导师,研究方向为数据库. E-mail: xqgong@sei.ecnu.edu.cn.

normalization to design the schema for OceanBase and to improve the performance of OLAP. Finally, we illustrate the efficiency and effectiveness of the denormalization design for OceanBase in the empirical study by using benchmark TPC-H.

Key words: denormalization; distributed database; OceanBase; TPC-H

0 引　　言

数据库按照表结构来组织、存储和管理数据，被广泛应用于各行各业，以提供数据支撑[1]. 数据库模型主要分为网状模型、层次模型、关系模型和面向对象模型等[2]. 其中，关系模型使用二维表来描述数据间的关系，具有很好的数据独立性和简洁的数据组织结构. 目前主流数据库管理系统均采用关系模型，如 Oracle[3]、DB2[4]、MySql[5]等，这类传统的关系型数据库采用集中式的管理模式，即数据的存储和处理一般都集中于一个节点上. 而数据库厂商则通过不断地提高节点的存储和处理能力，来应对业务和数据量的增长. 但是，由于近几年互联网的快速发展，全球数据量正呈爆炸式增长，传统的集中式数据库难以承载海量数据的存储和访问.

为解决这一问题，分布式数据库是一个不错的选择. 在分布式数据库中，系统通常会根据数据分片和数据分配的策略，将数据分布存储在多个节点上. 节点之间通过网络相互连接，当数据库收到数据请求时，能够将请求发往相应的节点进行处理，降低了单个节点的负载. 另外，在分布式数据库中，为了防止数据由于节点故障而造成丢失或无法访问，一般会为数据创建多个副本，分别存储在不同的节点上. 这样，即使有某个节点出现故障，也不会影响数据库的正常使用，很大程度上提高了数据库的可靠性. 分布式数据库很好地解决了集中数据库的数据存储瓶颈，并提升了数据库的可靠性. 但是在实际应用中，它仍存在一些问题. 在分布式数据库中，节点之间通过网络互连，数据会跨节点甚至跨地域分布. 如果应用中存在连接操作，并且涉及的各张表数据分别存储在不同节点上，那么执行该应用时将会造成数据的跨节点交互. 特别是对于 OLAP 业务，数据量一般十分庞大，连接操作导致的大量数据交互会产生巨大的网络传输开销，严重影响数据库的性能. 此外，由于数据分布存储并具有多个副本，副本间的数据一致性需要数据库来维护，这也会给数据库的性能带来负面影响.

在实践中，数据设计通常都会遵循范式. 然而，严格的范式设计无法保证数据库在任何情况下都能有最优的性能[6]. 为减少数据冗余，范式设计的数据库通常会包含大量的表格，因此，查询时可能需要连接多张表后才能获得需要的全部数据. 而过多的连接操作是数据库性能下降的主要因素，特别是在大数据量情况下，影响更大. 因此，通过减少表的数量，同时增加数据冗余，尽量减少查询中的连接操作，从而提高数据库性能，这正是所谓的反范式数据库模式设计. 尤其是在数据仓库业务中，由于数据量相对庞大，查询请求也较为复杂，反范式设计可以很好地改善数据库性能，以满足业务需求.

由于在分布式数据库中增加了网络传输开销，传统的数据库范式设计方法带来的问题被放大. 因此，反范式模式设计可能是分布式数据库的一个选择，这是因为：(1) 分布式数据库通过多节点数据冗余提高系统可用性，数据的完整性不再是通过减少冗余来实现；(2) 反范式模式设计降低了网络数据传输开销，可以提高分布式数据库的性能. 但是反范式模式设计也有缺点，比如在 OLTP 中，数据库的增、删、改的操作十分常见，较多的数据冗余会增加

操作异常、降低数据独立性;但 OLAP 的绝大部分是海量数据库上的 select 操作,不会导致操作异常,也不会影响数据独立性.因此,反范式模式设计是分布式数据库中 OLAP 业务的首选.本文主要针对 OLAP 介绍分布式数据库的模式设计方法,为 DBA 提供一些思路和方法.基于分布式数据库 OceanBase[7],通过常用的基准测试 TPC-H,验证了反范式模式设计在分布式 OLAP 业务上的有效性.

本文第 1 节介绍了传统数据库模式设计,包括范式设计和反范式设计,并列举了常用的反范式方法.第 2 节结合 OceanBase 数据库,分析如何使用反范式模式设计方法改善 OLAP 的性能.第 3 节通过实验验证了反范式模式设计在 OLAP 业务中的有效性.最后,第 4 节总结全文.

1 数据库模式设计

数据库设计一般要经过三个步骤:概念设计、逻辑设计和物理设计[8].概念设计,就是对真实世界中的各种实体进行提炼和抽象,并明确它们之间的关系,是现实世界到数据实体的第一层抽象,一般通过 E-R 图来描述.逻辑设计是针对特定数据库,根据概念设计阶段的数据实体和关系,建立一组数据库表结构的描述,包括确定表的属性、主键以及表之间的关联,它是数据库设计的关键步骤,决定了设计的优劣.物理设计将依据具体数据库的功能,确定表格各属性的存储类型和约束,最终得到所需要的数据库.

1.1 数据库范式设计

简单来说,范式设计是将数据属性进行组织、精炼的过程.E. F. Codd 最先开始了范式研究,并提出了三大范式[9,10].使用范式设计的数据库,可以合理地组织数据,使得数据库的各表之间具有很高的独立性,很大程度上减少了数据冗余.在进行更新、删除等操作时,由于数据独立性高,用户通常只需要对一张表的数据做相应操作,减少了由于数据冗余带来的各种操作异常.对于 OLTP 应用,例如在线交易、银行转账等,通常会存在大量的增、删、改等操作.在 OLTP 应用中使用范式能够有效组织数据,并减少数据冗余和更新异常.依照范式设计数据库,实现数据库规范化,对于数据库性能一般会有积极影响.范式设计虽然提高了数据独立性,但也意味着增加了表的数量,导致表间关联操作的增加,而这正是影响关系型数据库查询效率的关键所在.在 OLAP 业务中,绝大部分需求都是查询统计.这些查询需求一般都比较复杂,涉及到很多的统计分析和表的连接操作.而且由于 OLAP 业务需要处理的数据量十分庞大,数据库的性能必将因为大量的连接操作而大幅下降,以至无法满足业务需求.

1.2 数据库反范式设计

为降低查询复杂度,提高查询效率,针对 OLAP 业务,在数据库的模式设计阶段反范式(Denormalization) 设计是一种重要的方法.

所谓反范式设计,是指通过降低传统的数据规范化程度,以提高查询性能的过程.通过反范式可以显著减少表的数量,并减少查询请求中的连接操作的数量.目前,已经有许多关于反范式设计的应用和方法的研究.Tupper[11] 提出两种方法,分别在数据库设计的不同阶段考虑反范式,一种方式是在设计 ERD(实体联系图)阶段,就尽可能减少逻辑实体的数量,不用过于考虑数据的冗余度;另一种方式则是在物理设计阶段,针对具体的应用,使用反范式的方法增加表或表的属性,引入数据冗余并通过触发器来保证冗余数据的更新一致性.

Date[12]认为，应当在物理设计时使用反范式，而不建议在逻辑设计阶段考虑反范式，否则可能会影响原本清晰合理的数据结构和逻辑关系；Hahnke[13]证明了在商业分析型的应用中，反范式设计的数据库模式可以提供更佳的查询和分析性能. Rodgers[8]讨论了反范式的应用场景，介绍了何时应考虑进行反范式设计，并认为通常在一对一的实体关系中，反范式是有效的. 反范式设计也有其不足之处，例如数据冗余、数据更新异常以及更复杂的数据完整性约束. Coleman[14]认为，反范式设计在适应数据变化上有所不足，需要设计者深刻理解业务场景，充分考虑数据更新的频繁性，如果数据经常进行更新，那么对于这些数据就需要慎用反范式方法，以免造成更新效率降低，甚至产生数据不一致. 尽管反范式设计有缺点，但合理的设计可以降低这些缺点带来的不利影响，充分发挥反范式的优点可以提高 OLAP 业务的性能.

1.3　数据库反范式设计

反范式会通过增加冗余来提高查询性能，即所谓的“以空间换时间”. 需要注意的是，反范式不是一种设计规范，而仅仅是当数据库性能不能满足要求时的一种解决方案. 通常都是先按照范式进行数据设计，然后基于当前数据组织结构使用反范式策略，以提高查询效率. 反范式设计通常有下列几种方法[15,16].

(1) 通过预连接减少连接开销. 如果用户需要经常使用包含多表连接操作的查询，可以提前将这些表连接后的结果存入一张单表中. 这样使得查询由对多表的操作转为单表，避免了重复连接，能显著提高效率. 当连接涉及的表的更新较少时，预连接结果可以长时间保持不变，因此这种方法是比较有效的.

(2) 通过表镜像提高访问并发度. 镜像技术主要对数据库中访问量较高的表进行复制，源表和镜像表的模式完全相同，均可以同时对外提供服务. 由于源表和镜像表之间的数据是异步的，源表和镜像表的数据可能会出现不一致，因此，源表可以提供对外的读和写服务，而镜像表一般只对外提供读服务.

(3) 生成统计报表提高 OLAP 查询效率. 很多业务是通过查询获得一系列报表数据，涉及大量的统计计算. 为了提高效率，可以事先将报表所需的信息进行统计，并记录到一张单表中. 由于报表通常是对过去一段时间信息的统计，对实时数据不敏感，因此可以在每日访问低峰期生成报表，用于第二天的查询.

(4) 拆分表结构减少用户查询时涉及的数据量. 根据业务查询需要的字段的不同，将表格拆分成多个子表. 每个业务只需要访问各自的子表即可获取所需数据，减少了需要搜索的数据量. 拆分表结构一般包括垂直拆分和水平拆分，垂直拆分是将源表的属性根据不同的业务需求分割开，存入不同的新表中，每张新表都冗余原始表的主键属性，以保证记录唯一性；而水平拆分则是保留源表所有属性，根据主键范围进行拆分，拆分后每块数据存入新的表中，新表的结构与源表一致.

(5) 增加派生属性减少查询时的运算开销. 如果有查询需要经常对表中某几个属性数据做相同运算，那么可以为表格增加一个派生属性，用于存储这几个属性数据预算的结果. 如果数据有更新，派生属性也需要做相应的更新. 之后该查询就可以直接从派生属性获取数据，无需再做运算.

(6) 通过冗余数据避免表连接，提高查询效率. 与预连接方法不同的是，冗余数据的方法是在逻辑设计时，就将原本需要连接多张表才能获取到的属性存储在一张表中. 查询所需

要的数据都可以在这张表中得到,也就不必再进行连接操作.

2 分布式数据库中模式设计分析

2.1 OceanBase 系统架构

近年来,分布式数据库产品也在商业实践中得到成功应用,例如 Oracle 公司的 Mysql Cluster[17],Google 的 Spanner[18],Greenplum[19] 以及淘宝的 OceanBase[7] 等. 其中,OceanBase 是淘宝自主研发的关系型分布式数据库,支持千亿级记录的跨行跨表事务,能够为互联网级应用提供数据存储平台,并满足其高性能并发访问需求. OceanBase 实现了数据的分布存储并提供了数据的基本操作功能. OceanBase 将数据划分后分布式地存储在不同的节点上,极大增加了数据库的存储能力. OceanBase 根据服务器节点功能的不同,将其分为 4 类,包括 RootServer、UpdateServer、ChunkServer 和 MergeServer,如图 1 所示. 其中 ChunkServer 负责数据存储,集群中大部分节点都用于部署 ChunkServer;MergeServer 主要用来做 SQL 解析和执行,并收集执行结果返回给用户,一般与 ChunkServer 共享物理节点;UpdateServer 是 OceanBase 中唯一接收写请求的节点,写入的数据会定期合并到 ChunkServer 中;RootServer 主要用于管理集群中各个节点,维护数据分布信息,以及其它一些全局参数.

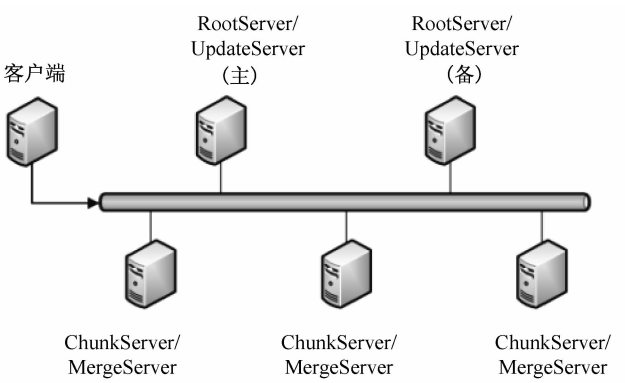

图 1 OceanBase 物理架构[7]

Fig. 1 The physical architecture of OceanBase

OceanBase 的查询处理流程与其他分布式数据库有所区别. 当查询开始时,MergeServer 会接受并解析查询请求. 由于 OceanBase 自动对表进行水平切分,每各分块可能存储在不同的 ChunkServer 上,因此 MergeServer 可能会将数据请求发往不同 ChunkServer. 最终 MergeServer 再将从各个存储节点上获取的数据集进行合并,并返回给用户. 对于涉及表连接的查询,数据库会首先根据查询条件对各个关联表过滤,这个过程相当于对每张表做单表查询,之后 MergeServer 会将返回结果进行连接等操作,最后将结果返回给用户. 整个查询处理流程中,会包含多次网络传输;如果 ChunkServer 返回的结果集过大,会导致很大的网络开销. 另外,数据处理和连接操作的压力都集中在 MergeServer,使得 MergeServer 负载较高. 为优化性能,OceanBase 中引入了主键序和主键前缀的概念. 在定义表的模式时,可以按序将表中的若干个字段,如 c1、c2,定义为联合主键,c1 和 c2 分别被称为第一主键和第二

主键. 其中 c1 的主键序为 1，而 c2 的主键序为 2. 第一主键 c1 被称作 c1、c2 联合主键的主键前缀. OceanBase 针对主键前缀做了索引优化，因此应用中使用主键前缀进行查询，可以显著提高查询效率.

OceanBase 适用于海量数据的存储并能够应对网络级数据应用，并具有良好的可扩展性，应用前景广泛. 目前，OceanBase 已经在淘宝业务中得到广泛使用，成功的实践证明了 OceanBase 的可靠性. 但是，由于 OceanBase 的开发初衷是支持淘宝业务，其功能还不如 DB2 等传统数据库齐全，例如不支持 where 条件的子查询、二级索引等功能，也不支持 DATE、CLOB、BLOB 等数据类型. 另外，OceanBase 系统虽然解决了数据存储的问题，但在处理 OLAP 业务时的性能仍有待提高. 其实，由于网络开销较大，OceanBase 都无法高效地处理 OLAP 业务. 下面几节，本文将结合 OceanBase 的架构，介绍如何在分布式数据库中合理使用反范式来提高 OLAP 业务的查询效率，并使用 TPC-H 作为测试基准进行验证.

2.2　TPC-H 案例研究与分析

TPC-H[20] 是 TPC（Transaction Processing Performance Council，事务处理性能委员会）提出的基准程序，是一个决策支持的标准，常用于对 OLAP 数据库性能进行基准测试. TPC-H 包含一组面向商务应用的查询语句和数据修改语句，共 22 个，这些语句都具有代表性且易于实现. TPC-H 中共有 8 张基本表，表的模式（Schema）定义和各个表的关系如图 2 所示.

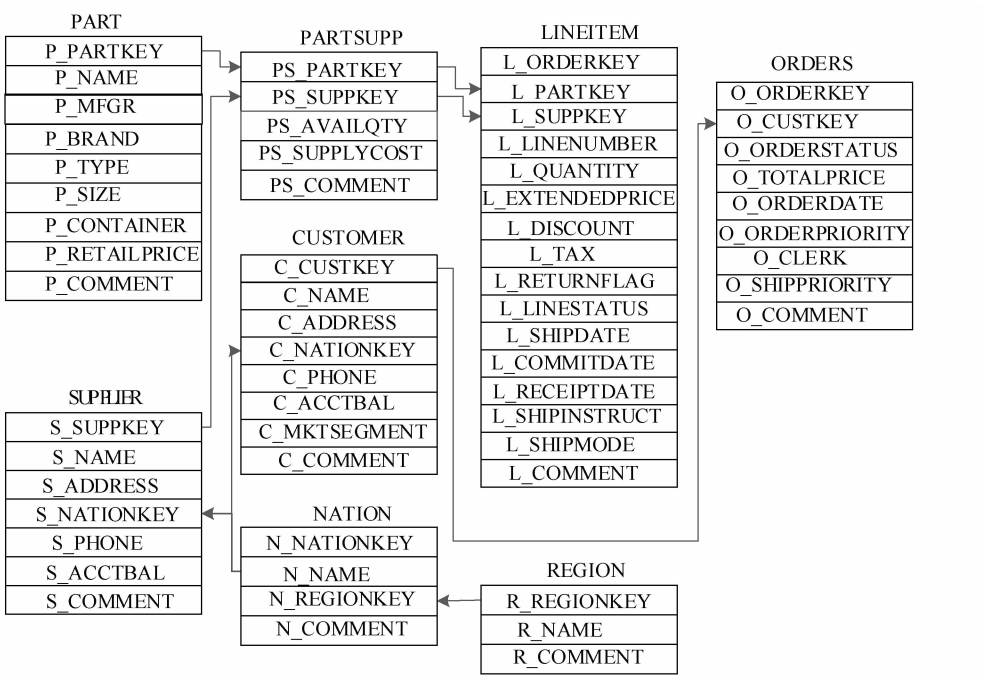

图 2　TPC-H 模式

Fig. 2　The Schema of TPC-H

TPC-H 包含的 22 个查询测试案例中大部分都比较复杂，具有随机特性. 表 1 列出了 22 个查询案例，并指出了每个查询案例中对性能影响最大的因素，并列出各个案例涉及到的表格. 少数案例只涉及单表查询，但包含了大量的统计计算，这样会严重影响查询效率；而

多数案例中存在多表连接,不仅包括显式的 JOIN 语句,还包含隐含在子查询中的连接.经过分析发现,国家表 nation 和地域表 region 在很多案例的 SQL 中都被关联在一起,而且这两张表的数据通常不会被更新和修改.因此,可以将 nation 表与 region 表合并为一张新的表 nation_re,这样就避免了这两张表的连接操作.orders 表存储了订单信息,很多 TPC-H 的案例,例如 Q2、Q3 和 Q4 等,希望同时获取交易明细信息(lineitem)、客户信息(customer)和订单信息(orders),或者三者之二.lineitem 和 orders 的数据量都很大,又经常相互连接.为进一步减少表连接,特别是大表连接操作,可以在 orders 表中冗余 lineitem 和 customer 表的信息,得到表 orders_li_cu.对于查询 Q1 来说,复杂的统计计算严重影响了查询效率.可以先通过 SQL 将需要获取的统计信息计算出来,存入统计表中,查询时只需要访问统计表即可.

表 1 TPC-H 查询案例情况

Tab.1 Cases in TPC-H

案例名称	影响性能的因素	涉及的表格
Q1 价格摘要报告查询	统计计算	Lineitem
Q2 最小代价供应者查询	多表连接	partsuppms,pplier,part,supplier,partsupp,nation,region
Q3 送优先权查询	多表连接	customer,orders,lineitem
Q4 订单优先权检查查询	多表连接	orders,lineitem
Q5 当地供应者数量查询	多表连接	customer,orders,lineitem,supplier,nation,region
Q6 当地供应者数量查询	统计计算	Lineitem
Q7 货运量查询	多表连接	supplier,lineitem,order,customer,nation,region
Q8 国家市场份额查询	多表连接	part,supplier,lineitem,orders,customer,nation,region
Q9 产品类型利润估量查询	多表连接	part,supplier,lineitem,nation,partsupp,orders
Q10 返回项目报告查询	多表连接	customer,orders,lineitem,nation
Q11 重要库存标志查询	多表连接	partsupp,supplier,nation
Q12 货运模式和命令优先查询	多表连接	orders,lineitem
Q13 消费者分配查询	多表连接	customer,orders
Q14 促进效果查询	多表连接	lineitem,part
Q15 促进效果查询头等	多表连接	supplier,lineitem
Q16 零件/供应商关系查询	多表连接	partsupp,part
Q17 小量订单收入查询	多表连接	lineitem,part
Q18 大订单顾客查询	多表连接	customer,orders,lineitem
Q19 折扣收入查询	多表连接	lineitem,part
Q20 潜在零件促进查询	多表连接	supplier,nation,partsupp,part,lineitem
Q21 不能按时交货供应商查询	多表连接	supplier,lineitem,orders,nation
Q22 全球销售机会查询	多表链接	customer,orders

由于 OceanBase 支持的功能有限,凡涉及视图、exist 语句、where 条件的子句等功能都无法在 TPC-H 上进行测试.经过筛选,在 TPC-H 的 22 查询案例中 OceanBase 只支持价格摘要报告查询(Q1)和运送优先权查询(Q3).其中,Q1 是单表查询,但是涉及到了大量统计计算,而 Q3 涉及多表连接.这两个案例在 SQL 语法上较为简单,在逻辑上易于理解,而且不涉及 OceanBase 不支持的功能,能够很容易地迁移到 OceanBase 上进行实验.另外,Q1 和 Q3 都是典型的 OLAP 查询,它们包含的数据统计和多表连接也都是传统数据库中常见的.如果使用严格的范式数据库设计,其查询效率通常不高.

价格摘要报告查询 Q1 提供了给定日期运送的所有行的价格摘要报告.每次查询时,系统都会将符合条件的数据记录全部从 ChunkServer 迁移到 MergeServer 上,然后由

MergeServer 完成统计计算，这使得 MergeServer 负载很高，查询响应时间长. 如果有大量用户同时提交该查询，甚至会影响整个数据库的访问效率. 因此，本文根据反范式模式设计策略，预先生成统计报表 lineitem_sta，其中，Q1 中作为分组条件的两个字段 l_retrunflag 和 l_linestatus，以及运送日期 l_shipdate 组成联合主键，l_shipdate 设为第一主键. 统计表还包含 sum_qty，sum_base_price，sum_disc_pirce 等统计属性. 统计报表中的每条记录表示某个运送日期之前(包含运送日期)，某组 l_returnflag 和 l_linestatus 的统计信息. 这样，查询 Q1 可以直接根据 l_shipdate 查询 lineitem_sta 的若干条记录获取所需结果.

运送优先权查询 Q3 给出在指定的日期之前尚未运送的订单中具有最大收入的订单的优先权和潜在的收入. Q3 需要连接三张表，其中 lineitem 表和 orders 表的数据量都很大. 并且查询语句中给出的查询条件都是范围查询，使得每张表过滤出的结果集较大，导致大量数据从 ChunkServer 通过网络传输到 MergeServer，造成很大的网络开销. 另外，将几个大数据量的结果集进行连接的效率也是很低的. 经过分析，可以对 orders、lineitem 和 customer 合并后的表 orders_li_cu 直接做单表查询.

最后，经过反范式设计的新 TPC-H 模式如图 3 所示. 新的模式中，orders_li_cu 表冗余存储了 orders、lineitem 表和 customer 表的所有数据，去除了原来的三张表. 将 nation 表和 region 表合并为 nation_re. 另外，还增加了 lineitem 的统计表 lineitem_sta.

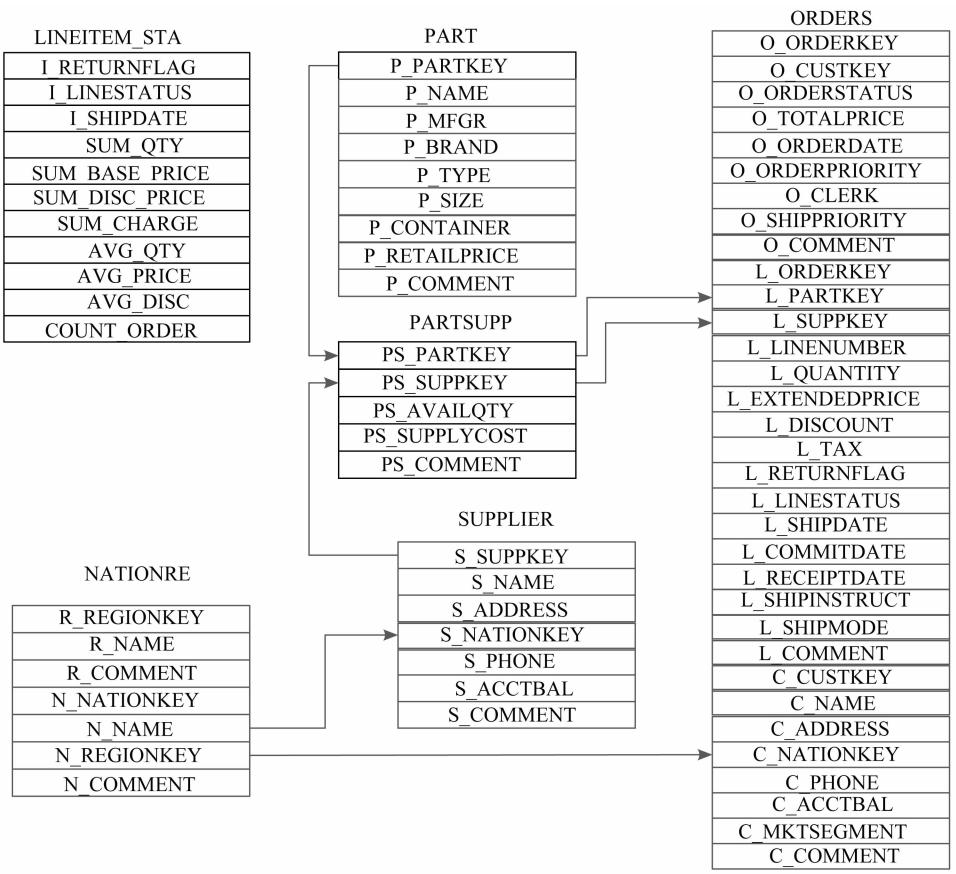

图 3　新 TPC-H 模式

Fig. 3　New schema of TPC-H

3 实 验

根据新的数据库模式,将 Q1 和 Q3 对应的 SQL 查询需要做相应调整. Q1 可以使用图 4(右)所示的查询 SQL 代替. 其中,为了方便描述,将 SQL 语句中的条件 date '1998-12-01'-interval '[DELTA]'day (3)直接改为'[DATE]',表示在[DATE]这个日期之前的数据需要进行统计.

```
select
l_returnflag
l_linestatus,
sum(l_quantity)
as sum_qty,
sum(l_extendedprice)
as sum_base_price
sum(l_extendedprice*(1-l_discount))
as sum_disc_price,
sum (l_extendedprice*(1-l_discount)*(1+l_tax))
as sum_charge,
avg(l_quantity)
as avg_qty,
avg(l_extendedprice)as avg_price,
avg(l_discount)as avg_disc,
count(*)as count_order
from lineitem
where l_shipdate<=date'1998-12-01'
-interval'[DELTA]'day(3)
group by l_returnflag,l_linestatus
order by l_returnflag,l_linestatus;
```

⇨

```
select
l_returnflag,
l_linestatus,
sum_qty,
sum_base_price,
sum_disc_price,sum_charge,
avg_qty,
avg_price,
avg_disc,
count_order
from lineitem_sta
where l_shipdate='[DATE]'
order by l_returnflag,l_linestatus;
```

图 4 使用反范式设计后 Q1 查询 SQL 的修改

Fig. 4 Modification of SQL query for Q1 after denormalization

由于在新模式中,表 orders_li_cu 已经包含了原模式 orders, lineitem 和 customer 中所有的字段,因此 Q3 就再不需要进行 join 连接,而只需从 oreders_li_cu 中获取数据. 因为 l_orderkey = o_orderkey 且 c_custkey = o_custkey,所以在扩展表 orders_li_cu 中只需要 o_custkey,不必冗余 l_orderkey 和 custkey 字段. 查询案例中可以去除条件 l_orderkey = o_orderkey,并将 l_orderkey 改为 o_orderkey. 同理,也可以去除条件 c_custkey = o_custkey. 查询 SQL 调整后的结果见图 5(右).

```
l_orderkey,
sum(l_extendedprice*(1-l_discount))as revenue,
o_orderdate,
o_shippriority
from
customer,orders,lineitem
where
c_mktsegment='[SEGMENT]'
and c_custkey=o_custkey
and l_orderkey=o_orderkey
and o_orderdate<date'[DATE]'
and l_shipdate>date'[DATE]'
group by
l_orderkey,o_orderdate'o_shippriority
order by
revenue desc,o_orderdater;
```

⇨

```
select
o_orderkey,
sum(l_extendedprice*(1-l_discount)) as revenue,
o_orderdate,
o_shippriority
from orders _li_cu
where
c_mktsegment='[SEGMENT]'
and o_orderdate<'[DATE]'
and l_shipdate>'[DATE]'
group by
l_orderkey,o_orderdate,o_shippriority
order by
reveue desc,o_orderdate;
```

图 5 使用反范式设计后 Q3 查询 SQL 的修改

Fig. 5 Modification of SQL query for Q3 after denormalization

在 Linux 服务器上搭建 0.4.2.18_22M 版本的 OceanBase 数据库，服务器的硬件配置说明见表 2.

表 2　服务器的硬件配置

Tab.2　The configuration of servers

Linux 版本	Red Hat Enterprise Linux Server release 6.2(Santiago)
CPU	Inter(R) Xeon(R) CPU E7-4860，2.27 Ghz，6 核
内存	50 GB
硬盘	SATA 硬盘，100 GB 以上

在 OceanBase 中新建数据表后，使用 TPC-H 数据生成器生成数据(SF 参数设置为 1)并进行数据加载. 由于只测试 Q1 和 Q3 两个案例，因此只需加载 orders、customer 和 lineitem 三张表，其记录数分别为 150 000、150 000 和 6 001 215.

当测试 Q1 时，选取 6 个[DATE]参数进行查询，与 TPC-H 原始案例进行比较. 测试 Q3 时，[SEGMENT]设置为 BUILDING，[DATE]参数则在[1995-03-01，1995-03-31]中选取，共取 6 个日期作为参数，间隔 5 d. 实验结果如表 3 与表 4 所示. 表 3 中，随 DATE 参数增长，根据 Q1 中的过滤条件，查询时需要进行统计的数据集也随之增大，使得使用旧模式设计的数据库响应时间成线性增长. 但使用反范式重新设计模式后，表中只需存储 Q1 的统计结果，无需进行复杂计算，降低了系统开销. 查询时，通过 DATE 参数，可以直接获得该日期之前的统计信息，因此新的数据库模式下，查询响应时间得到显著提高，且基本不会随 DATE 参数的变化而改变. 表 4 是 Q3 的实验结果，可以看出查询效率得到了很大提升，基本相差一个量级. 这是由于重新设计的数据库模式中，已经将查询 Q3 所涉及的字段值都冗余到了表 orders_li_cu 中，将原来的三表连接查询改为单表查询，避免了连接操作，减少了网络传输的开销.

表 3　查询 Q1 的响应时间

Tab.3　The response time of Q1

DATE	1993-01-01	1994-01-01	1995-01-01	1996-01-01	1997-01-01	1998-01-01
范式模式响应时间/s	23.73	42.64	60.87	78.49	97.17	117.93
反范式模式响应时间/s	0.01	0.01	0.01	0.01	0.01	0.01

表 4　查询 Q3 的响应时间

Tab.4　The response time of Q3

DATE	1993-01-01	1994-01-01	1995-01-01	1996-01-01	1997-01-01	1998-01-01
范式模式响应时间/s	72.21	72.02	69.98	71.14	71.31	72.12
反范式模式响应时间/s	6.75	6.92	6.74	7.04	7.02	6.81

4　总　　结

本文介绍了范式设计和反范式设计的区别和应用场景，以及一些典型的反范式模式设计方法，然后通过重新设计 TPC-H 基准测试案例中的 Q1 和 Q3 相关数据库模式，在分布式数据库 OceanBase 中对查询效率的提升进行了实验验证. 结果表明，新的数据库模式可以有效提升 OLAP 业务的查询效率. 虽然增加了数据冗余，但是考虑分布式环境下的存储容

量不再是瓶颈,数据冗余可以不作为考虑因素.在进行分布式数据库模式设计时,需要充分考虑业务需求,适当利用一些反范式方法,可以有效优化查询性能.

[参 考 文 献]

[1] SILBERSCHATZ A, KORTH H F, SUDARSHAN S. 数据库系统概念[M]. 杨冬青,马秀莉,译. 北京:机械工业出版社,2012.

[2] 王珊,萨师煊. 数据库系统概论[M]. 北京: 高等教育出版社,2006.

[3] Oracle[DB/OL]. http://www.oracle.com/index.html.

[4] BD2[DB/OL]. http://www-01.ibm.com/software/data/db2.

[5] Mysql[DB/OL]. www.mysql.com.

[6] RODGERS U. Denormalization: why, what, and how? [J]. Database Programming and Design,1989,2(12):46-53.

[7] 杨传辉. 大规模分布式存储系统:原理解析与架构实战[M]. 北京: 机械工业出版社,2013.

[8] STEPHENS R K, PLEW R R. 数据库系统概论[M]. 何玉洁,译. 北京: 机械工业出版社,2001.

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

[10] CODD E F. Normalized data base structure: A brief tutorial[C]//Proceedings of ACM SIG-FIDET Workshop on Data Description, Access, and Control. San Diego, California, Nov. 11-12,1971.

[11] TUPPER C. The physics of logical modeling. Database[J]. Programming & Design, 1998.

[12] DATE C J. The birth of the relational model[J]. Intelligent Enterprise Magazine. 1998.

[13] HAHNKE J. Data model design for business analysis[J]. Unix Review, 1996,14(10).

[14] COLEMAN G. Normalizing not only way[J]. Computerworld, 1989: 63-64.

[15] HOLLINGSWORTH M. Data normalization, denormalization, and the forces of darkness[EB/OL]. http://www.fastanimals.com/melissa.

[16] SANDERS G L, SHIN S. Denormalization effects on performance of RDBMS[C]//Proceeding of the 34th Hawaii International Conference on System Sciences-HICSS,2001.

[17] MySQL Cluster. Overview[EB/OL]. http://dev.mysql.com/doc/refman/5.0/en/mysql-cluster-overview.html.

[18] CORBETT J C, DEAN J, EPSTEIN M, et al. Spanner: Google's Globally-Distributed Database[C]//Proceedings of the first USENIX Symposium on Operating Systems Design and Implementation (OSDI),2012.

[19] GREENPLUM 数据库引擎探究[DB/OL]. http://www.cnblogs.com/daduxiong/archive/2010/10/13/1850411.html.

[20] Transaction Processing Performance Council (TPC). TPC BENCHMARKTM H [EB/OL]. (1993)[2013-12-01]. http://www.tpc.org/tpch/spec/tpch2.17.0.pdf.

(责任编辑 王善平)