论文摘要

联机分析处理（OLAP）是当前数据仓库应用和决策支持系统（DSS）的研究热点。OLAP 查询通常需在海量数据上进行即席（ad hoc）的复杂聚集查询，并要求及时向用户提供分析数据，用以辅助决策。这种使用方式对查询响应速度提出了很高的要求，使得提高 OLAP 查询和分析操作效率成为数据仓库应用中的关键问题。

本文是在参与研究和开发一个数据仓库系统（包括数据仓库的建模、ETL、OLAP 系统以及应用工具等）的基础上，着重对提高 OLAP 查询分析效率的 ROLAP 聚集技术、聚集 Cube、实视图的选择与分割及其增量更新维护等 OLAP 若干关键技术进行了系统深入的研究。本文的主要研究工作及其所取得的创造性成果有：

（1）研究了维层次编码，提出一种基于维层次编码的新型预分组聚集算法 DHEPGA。DHEPGA 算法充分利用了编码长度较小的维层次编码及其维层次前缀路径，通过维层次编码前缀匹配操作，快速检索出与查询关键字相匹配的维层次编码，求得维层次属性的查询范围，大大减少和简化事实表与维表之间的多表连接，减少了 I/O 开销，提高了 OLAP 查询效率。

（2）利用 Cube 中的维层次聚集技术实现高性能的维层次聚集 Cube（DHAC）。在 DHAC 中进行数据插入和删除等数据更新时，充分利用维层次聚集树中的维层次前缀，对受到更新影响的所有祖先结点进行增量更新。在插入新维数据时，不需要重新构建聚集 Cube 就可以对 DHAC 进行增量更新，从而提高了 Cube 的更新效率。

（3）提出了一种数据仓库实视图选择和分割算法。根据 OLAP 查询中的选择谓词构造的最小项谓词，选择数据仓库视图进行有效的水平分割和实体化，使 OLAP 查询通过访问较少元组的分割裂片就可以完成，从而加快了 OLAP 查询响应时间，削减维护费用，提高了 OLAP 查询效率。

（4）提出了一种数据仓库实视图增量更新维护的有效策略。利用视图表达式树及其计算的中间结果来创建辅助视图，并进行实体化，从而实现数据仓库视图增量更新维护，有效地改善了 OLAP 查询的数据质量和效率。

关键字:联机分析处理;维层次编码;DHEPGA 聚集算法;维层次聚集 Cube;实视图;视图增量维护

ABSTRACT

Online analytical processing (OLAP) is a hot issue in the research area of the data warehouse and decision support system(DSS). The OLAP queries are ad hoc, complex aggregation queries on the massive data set. OLAP queries are complex and volume of data in large such that they make query response time and analytical efficiency as important issues in data warehouse.

In this paper it is emphasized how to improve the efficiency of OLAP queries with some key technologies of OLAP such as aggregate technology of ROLAP, aggregate cube, materialized view selecting and partitioning, and view incremental maintenance. The research is part of the project — "The research and implementation of the data warehouse system" to implement a proto type of data warehouse system named as SEUDW . The main contributions and innovations of this dissertation are as follows:

(1) This paper proposes a novel pre-grouping aggregation algorithm, DHEPGA (pre-grouping aggregation based on the dimension hierarchical encoding) . By using the small dimension hierarchical encoding and their hierarchical prefix path, DHEPGA can rapidly retrieve the matching dimension hierarchical encoding and evaluate the set of query ranges for each dimension. As a result, this algorithm can greatly reduce multi-table join, reduce the disk I/Os, and highly improve the efficiency of OLAP queries.

(2)In this paper it is researched how to create the highly performance dimension hierarchy aggregate cube (DHAC) with the dimension hierarchy aggregate technique on the cube. By using the hierarchical prefix of the dimension hierarchy aggregate tree, the DHAC can incrementally update affected ancestor while updating the data cell in it. The DHAC can also incrementally update without being recreated while being added new dimension data in it. As a result, this algorithm can highly improve the update efficiency.

(3) This paper proposes a method for selecting and efficiently horizontally partitioning warehouse views based on the minterm predicates of OLAP queries. So the OLAP queries can scan fewer fragments and scan fewer rows then all in the original view, and improve the efficiency of the OLAP queries.

(4) In this paper we show the warehouse views can be made incrementally maintainable with the auxiliary views , which derived from the intermediate results of the view computation can be materialized in the warehouse. This paper also proposes a method to compute the changes to each intermediate node in a bottom-up fashion in the view expression tree.

Key words: online analytical processing; dimension hierarchical encoding; DHEPGA aggregation algorithm;dimension hierarchy aggregate cube; materialized view; view incremental maintenance

Y 644284

东南大学学位论文独创性声明

本人声明所呈交的学位论文是我个人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得东南大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

研究生签名：胡弘信 日 期：2004.2

东南大学学位论文使用授权声明

东南大学、中国科学技术信息研究所、国家图书馆有权保留本人所送交学位论文的复印件和电子文档，可以采用影印、缩印或其他复制手段保存论文。本人电子文档的内容和纸质论文的内容相一致。除在保密期内的保密论文外，允许论文被查阅和借阅，可以公布（包括刊登）论文的全部或部分内容。论文的公布（包括刊登）授权东南大学研究生院办理。

研究生签名：胡弘信 导师签名：董达生 日 期：2004.2.

第一章 前 言

1.1 研究背景

随着信息处理技术的不断发展以及数据库管理系统（Database Management System，DBMS）的广泛应用，各行各业已建立了很多不同的计算机信息系统，积累了大量的数据。随着时间的增长和业务的不断增多，这些数据量和规模也急剧增长。人们正面临着数据的海洋，决策者如何从中得到有用的决策信息已成为一件难事。在目前日益激烈的企业竞争环境中，要求企业决策更加科学化，使企业决策层越来越意识到科学决策的重要性，而科学决策的关键是信息。数据仓库技术正是专门用于为决策者提供决策支持信息的。这种技术是根据决策要求将企业里多种不同计算机系统中的有关数据集成起来，存储到数据仓库系统（Data Warehouse，简称 DW）中，通过对数据仓库中的数据的查询、分析和挖掘，发现对决策有用的信息。

利用 OLAP(online analytical processing)强大的查询能力、数据对比和报表功能可以进行探测式数据分析，从而使得 OLAP 在数据仓库系统中发挥了关键性的作用[1-7]。OLAP 是从数据仓库中的综合数据出发，提供面向分析的多维模型，使用多维分析方法，从多个角度、多个侧面以及多个层次对数据进行分析与比较，使用户能够以更自然的方式获得丰富的信息。由于 OLAP 常常用来管理决策所需要的总结数据，数据量巨大，而且往往还需要满足用户的即席（ad hoc）查询，及时向用户提供分析数据，从而对查询响应速度提出了很高的要求，使得如何提高 OLAP 查询和分析操作效率成为数据仓库应用中的关键问题。OLAP 主要有 ROLAP（relational OLAP）和 MOLAP（multidimensional OLAP）两种具体形式。ROLAP 在大数据量的存储处理上有绝对的优势，MOLAP 在响应速度、预聚集和多维计算方面具有优势，因而通常利用 ROLAP 来存储明细数据，利用 MOLAP 来存储用于决策分析的多维聚集数据。

目前提高 OLAP 查询和分析操作效率的重要课题主要有：

（1）在 ROLAP 中查询往往需在海量数据上进行即席的复杂分组聚集计算，在其 SQL 语句中通常包含多表连接和分组聚集操作，因而使减少多表连接和压缩关键字，对查询数据进行有效地分组聚集操作，来提高 OLAP 查询效率，成为 ROLAP 的关键问题。

（2）在 MOLAP 中，主要是通过对 Cube 进行预聚集计算，将结果保存起来，来降低从头开始计算聚集值的费用，提高 OLAP 查询响应速度。但在源数据 Cube 某个单元数据更新时，对经过预聚集处理的聚集 Cube 中受到更新影响的所有单元都要做出相应的增量更新。另外，多维数据集合通常十分稀疏，使得 Cube 聚集技术、Cube 更新技术和多维存储技术成为数据仓库中的重要研究课题。

（3）在数据仓库中，为了提高 OLAP 查询响应速度，通常将存储在基表中的原始数据进行聚集计算，以实视图形式进行存储，造成在数据仓库中实视图可能达到几千个，给查询和更新维护带来了很大的的难度。从而使得如何适当地选择数据仓库视图并进行有效地分割、如何提高实视图增量更新效率也成了 OLAP 的一个关键问题，它将直接影响 OLAP 查询效率和数据质量。

（4）利用数据网格、分布式处理、并行算法等新技术，来对 Cube 进行聚集计算，以及对视图进行增量更新维护，来提高 OLAP 查询分析效率，改善 OLAP 查询的数据质量。

本文将主要针对以上这些课题中的 OLAP 聚集技术、实视图的选择与分割以及实视图的增量更新维护等关键技术展开较为深入的研究。

1.2 研究现状

1.2.1 ROLAP 聚集技术

在数据仓库中，OLAP 往往需在海量数据上进行复杂的分组聚集查询，为了提高查询响应速度，研究人员提出了索引技术[8-13]、实视图技术[14-16]、聚集查询的并行处理和查询优化[17-30]等方法和技术。但在 ROLAP 查询中通常包含多表连接和分组聚集操作，提高这些操作的性能成为提高 OLAP 查询响应速度的关键。目前已提出了一些新的分组聚集计算技术，如文献[31-35]等。在文献[31,32]中作者提出一种基于分组序号的聚集算法 MuGA，将聚集操作和多表连接算法 MJoin（multi-table join）相结合，使用分组序号对事实表中的数据进行分组聚集计算。该方法首先要计算出各个维表中每一个分组字段取值对应的分组序号，因而这些分组序号将随着各个维表分组方法的不同会重新计算分组序号，而且这些分组序号没有表达维属性的层次语义。这种方法最主要的缺点就是没有充分考虑维属性具有层次性这种语义特性，没有充分利用维层次特性对事实数据进行快速的分组聚集计算；也没有充分利用维层次编码前缀，来进一步提高分组聚集的操作效率。

1.2.2 聚集 Cube

在[36]引入 Data Cube 模型后，在数据库领域，专家学者先后在数据仓库中的 Cube 计算[37-52]、Cube 预聚集处理[53-59]、Cube 压缩存储[60-68]、语义 Cube[69,70]、Cube 更新[71-73]等方面取得了一些研究成果，提高了 Cube 的性能。其中，Ho 等人在文献[53]提出采用 PS(Prefix Sum)方法对 Data Cube 上数据进行预先计算其聚集值，PS Cube 的基本思想是存储预计算的聚集信息，从而在很短时间内回答即席查询。但是这种算法在 Data Cube 某数据单元更新时，处于该单元之后的所有数据单元都需进行更新，从而产生串联更新，在最坏情况下，将需对整个 Cube 都进行更新。为了解决 PS 中的串联更新问题，Geffner 等人又提出 RPS(Relative Prefix Sum)方法[54]和 DDC(Dynamic Data Cube)方法[55]，Liang 等人提出 Double RPS 方法[56]等，在一定程度上降低了数据更新费用及更新时间。但是这些方法需要大量的额外空间，为了降低存储空间及其额外存储费用，减少额外地更新的传播费用和物理磁盘延迟访问时间，Riedewald 等人提出 SRPS(Space-Efficient Relative Prefix Sum)方法和 SDDC(Space-Efficient Dynamic Data Cube)方法[57]，这两种 Cube 是在 RPS 和 DDC 进行了改进。作者在文献[62]中又提出了一种层次式 Cube（Hierarchical Data Cube,HDC）存储结构，使得区域查询的代价和数据更新代价均为 $O(log_2^d n)$（d 为多维数据集的维度，n 为维度的值域）。与以往传统的 Cube 相比，SDDC、HDC 在性能上有较大的提高。但这些方法都还未能更好地解决聚集 Cube 的高效存储和增量更新问题：

（1）未能解决多维数组存储的大容量和高稀疏度问题。采用多维数组存储时，数组过于稀疏，会导致大量的存储空间的浪费。以往均采用多维数组线性化来解决稀疏问题，但这需要进行大量的乘法和加法运算，计算工作量较大，给 OLAP 查询处理带来了额外的开销。

（2）未能有效地解决 Cube 中维层次结构的聚集操作和 OLAP 操作处理的效率问题。现有的多维数组存储结构没有保存维的层次及其层次关系等语义信息，不能支持 Cube 的 drill down 或 roll up 等语义操作。

（3）未能更好地解决在插入新维数据或数据单元时进行 Cube 的增量更新。在以往方法构造的多维数组存储方式中，在新的维数据或数据单元等增量数据插入时，常常需要重构多维数组，不能对多维数组进行增量更新。尤其是对 Cube 进行维以及维层次的增删等模式更新时，将引起整个 Cube 的重构；随着多维数组的增大，其重构的开销将急剧增加。

1.2.3 实视图选择与分割及增量更新维护

OLAP 查询常常是非常复杂的，需要涉及的数据量大，因而查询响应时间和维护代价是数据仓库应用中一个十分突出的问题。为了加快大量数据的查询处理，通常以实视图形式来存储由基表中的原始数据计算出来的用于决策的各种总结数据，在需要时只要读出，无需重新计算[74]。同时采用查询重写与优化[75-79]等技术，来提高 OLAP 查询响应速度。

1.2.3.1 实视图选择与分割

在数据仓库中，实视图可能达到几千个，这给查询和维护带来了很大的难度，从而使实视图的选择与分割[80-83]成为数据仓库中利用实视图技术对 OLAP 进行聚集查询的一个关键技术。在视图分割方面，Chaudhuri 等人利用垂直分割的想法来降低索引的存储和维护费用[83]。Noamn 等人提出分布式 DW 构建技术[84]，但他们都没有提及水平分割的问题。Ozsu 和 Valduriez 在[85]提出利用简单谓词对关系数据库进行水平分割的思想和方案，但他们均没有考虑查询访问频率。Bellatreche 等人提出对数据仓库星形模式采用分割技术来减少查询执行总费用[86,87]，但都是利用简单谓词子句对数据仓库中的事实表和维表进行分割。但这些分割方法的 OLAP 查询效率都不太理想，都没有充分利用查询选择谓词及其访问效率来更好地选择数据仓库视图，进行有效地水平分割和实体化，以提高 OLAP 查询效率。

1.2.3.2 实视图增量更新维护

在数据仓库中，当基础数据源由于插入、删除和修改而引起数据更新时，实视图需要定时的或按需进行更新维护，以确保查询的数据质量和数据的准确性。保持视图随着数据源的变化而更新的过程称为视图维护，视图维护问题已被广泛讨论[88-97]。Zhuge 等人提出 Strobe 算法[88,89]，进行多数据源的数据仓库一致性维护。这个算法为了实现维护，须在数据仓库和数据源之间来回地传送查询及其结果。Agrawal 等人提出 Sweep 算法[90]，利用临时查询结果解决数据仓库中视图一致性维护出现的异常问题，而不采用 Strobe 算法中的补偿查询算法。但是这些算法在网络严重拥赛时，都会引起其性能大大降低。视图维护可以通过视图的定义重新计算或视图的增量维护来完成。实视图增量维护是指通过已存在的视图和基关系的变化，来计算实视图的更新数据，现有的视图维护技术为了维护实视图都需要直接访问基关系。由于基关系分布在不同的数据源上，直接访问它们是非常困难的，访问费用也非常巨大。一个理想的实视图增量更新维护应能在不需要直接去访问源数据的情况下就可以对数据仓库实视图进行动态的增量更新，这个问题已成为数据仓库研究中的一个热点问题[98-100]。以往这些方法都是利用源数据或视图的更新值对由其生成的所有视图进行更新，更新代价较高，都没有充分利用视图表达式及其辅助视图对实视图进行快速地增量更新，来实现数据仓库视图增量更新维护。

1.3 主要研究目标

作者企图针对以上分析中提到的问题，对数据仓库中 OLAP 若干关键技术展开较为深入的研究，以期达到以下目标：

(1) 在 ROLAP 中，利用维层次编码来对星形模式或雪花型模式进行改进，来减少和简化 OLAP 查询中的多表连接操作，减少 I/O 开销，提高 OLAP 查询速度。

(2) 在 MOLAP 中，利用维层次聚集技术来创建维层次聚集 Cube，实现 Cube 的压缩存储和增量更新等。

(3) 对实视图的选择与分割、实视图增量更新维护进行研究，提出在实视图的选择与分割及增

量更新维护方面比较理想的策略和方案。

（4）充分利用本文在 OLAP 若干关键技术方面的主要研究成果，为下一步实现一个高性能多维数据仓库系统打下坚实的理论基础。

1.4 主要研究内容

本论文是结合江苏省“十五”高科技项目——“数据仓库系统研究与开发”展开有关课题的研究。该项目的第一个目标是研制出一套数据仓库化的工具集，支持企业数据仓库建设和应用开发的全过程。这套工具集主要是由建模系统、ETL 系统、OLAP 系统以及应用工具集四个部分组成，其整体结构如图 1-1 所示。

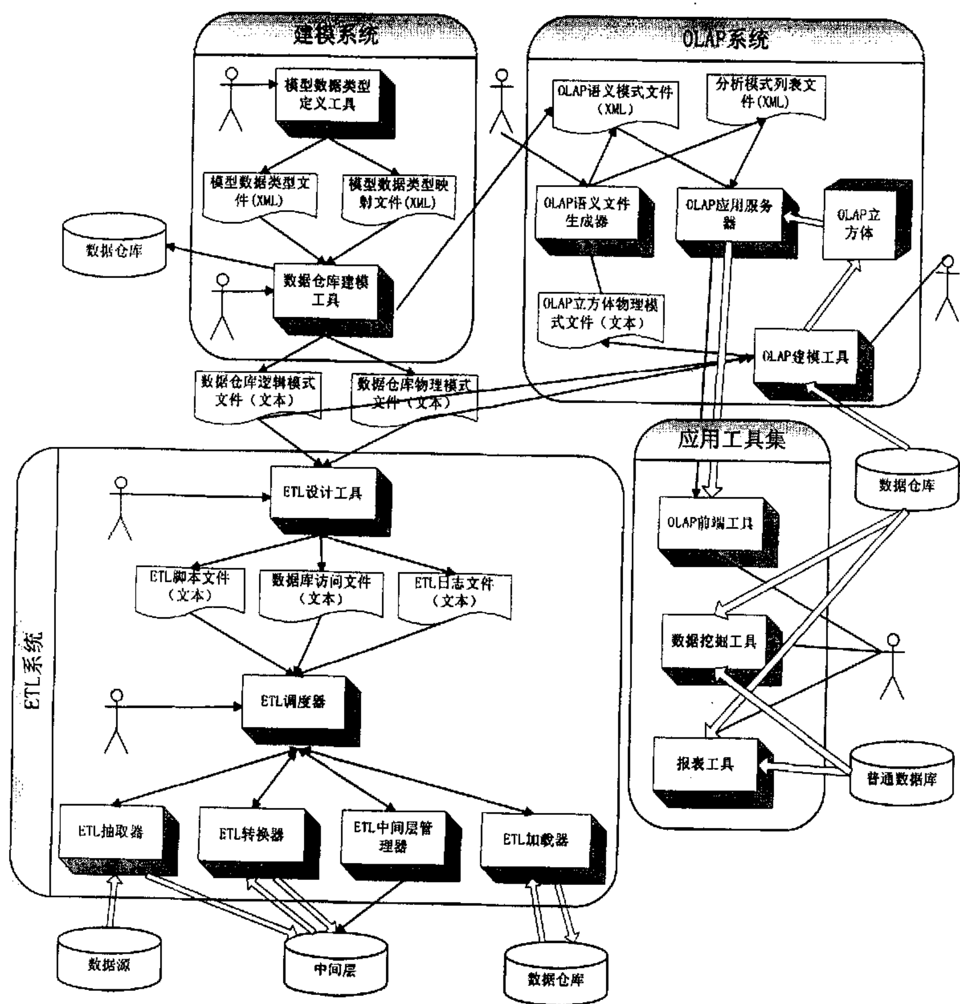


图 1-1 SEUDW 系统的整体结构

1）建模系统

建模系统是面向数据仓库设计人员，用来辅助设计企业级数据仓库。这个系统包括以下两个工

具：

（1）模型数据类型定义工具

可以定义数据仓库的数据类型，这些数据类型可以用于数据仓库模型的自动生成。该工具生成的数据类型定义文件可以使数据仓库建模脱离实际使用的数据库系统，在逻辑上定义，然后可以实现到不同类型的具体数据库系统上。

（2）数据仓库建模工具

数据仓库建模工具提供了一种星形设计模式，让用户在逻辑上设计多维分析模型，然后自动生成对应的物理数据模型，最后可以自动生成物理模型到实际的数据库系统中。

2）ETL 系统

ETL 系统是面向 ETL 设计人员和 ETL 管理人员，辅助完成 ETL 过程的设计和实现，半自动的进行数据的抽取、转换、加载，完成数据仓库中数据的加载和刷新。ETL 系统提供 ETL 设计工具、调度器、抽取/转换器、加载器、中间层管理器五个模块。其中 ETL 设计工具是提供 ETL 设计人员进行 ETL 过程设计，调度器、抽取/转换器、加载器、中间层合起来作为 ETL 过程执行系统，完成 ETL 设计工具生成的 ETL 过程定义。

（1）ETL 设计工具

ETL 设计工具允许用户以图形化的方式定义整个 ETL 过程中各个抽取、转换、加载等环节，能够半自动化地、快速简单地定义 ETL 过程，生成 ETL 过程元数据。

（2）调度器

调度器存储和维护 ETL 过程元数据，将 ETL 过程中各个任务发送到不同的执行模块上进行执行，同时维护和管理这些执行模块。提供一个灵活的机制让用户方便地查看和管理整个 ETL 过程。

（3）抽取/转换器

抽取/转换器执行各种数据转换任务，这些任务使用脚本来描述。通过解释执行脚本完成从数据源抽取数据，然后进行转换或者将中间层上的临时数据进行转换处理。

（4）加载器

加载器执行将数据从数据源或者中间层加载到数据仓库中星形模式中，动态生成星形模型的维表。加载器也是通过解释执行加载脚本来完成加载任务。

（5）中间层管理器

中间层管理器负责维护中间层上的临时数据表，通过执行中间层脚本自动完成中间层上数据表的清理。

3）OLAP 系统

OLAP 系统是面向分析设计人员和决策分析人员，分析设计人员为决策分析人员准备数据，决策分析人员利用准备好的数据进行分析得到结果。OLAP 系统提供 OLAP 建模工具、OLAP 语义文件生成工具、OLAP 应用服务器、OLAP 前端工具，其中前两个工具是面向分析设计人员进行数据的准备，后两个工具是面向决策分析人员，使用数据仓库中的数据进行在线分析。

（1）OLAP 建模工具

OLAP 建模工具用于设计 OLAP 立方体的数据模型，以及实现 OLAP 立方体的数据加载和数据更新。

（2）OLAP 语义文件生成工具

OLAP 语义文件生成工具为 OLAP 应用服务器提供用于分析使用的语义文件，使用数据仓库模式文件、OLAP 立方体模式文件直接生成对应的语义文件，或者手工定义针对普通关系数据库的语义文件。

（3）OLAP 应用服务器

OLAP 应用服务器接受 OLAP 前端的分析请求从不同的后台数据库中获取相关数据传输给 OLAP 前端，OLAP 应用服务器可以连接的后端包括多维数据库（SQL Server OLAP Service）或者

普通的关系数据库（数据仓库、一般的企业数据库）。

4）应用工具集

应用工具集包含了一组面向最终用户的分析工具，目前主要有 OLAP 前端分析工具、数据挖掘工具以及一个报表生成工具。这个工具集在结构上是开放的，可以很容易地接纳新的分析工具。

（1）OLAP 前端工具

OLAP 前端提供分析人员一个图形化分析界面，用户可以定义分析模式，然后在分析模式上进行上卷、下探、旋转和切片等多维分析，对于分析的结果可以生成打印报表或转为 Excel 格式，以便作其他后续处理。

（2）数据挖掘工具

目前提供的数据挖掘工具有：一个基于经典的 Apriori 算法的关联分析工具，可以分析数据项之间的关联关系；一个基于 ID3 决策树算法的分类分析工具，可以分析数据的类别分布；一个基于广度优先搜索邻居聚类算法的聚类分析工具，可以将数据按其特征进行聚类。

（3）报表工具

报表工具提供分析人员图形化的创建各种样式的自定义报表，采用语义对象技术屏蔽报表创建用户对数据库的了解。本报表工具可以按照模板创建普通报表，也可以直接定义报表格式，提供表中套表、画斜线、插入图象等功能。

本项目的第二个目标是结合数据仓库化工具的研制，对其中涉及的若干关键技术，主要用于提高 OLAP 查询和分析操作效率的 ROLAP 聚集技术、聚集 Cube、实视图的选择与分割及其增量更新维护等进行深入的研究，为系统进一步升级奠定基础，这也是本论文的主要研究内容。本文所研究的主要内容是以图 1-2 所示的多维数据仓库原型系统为背景。

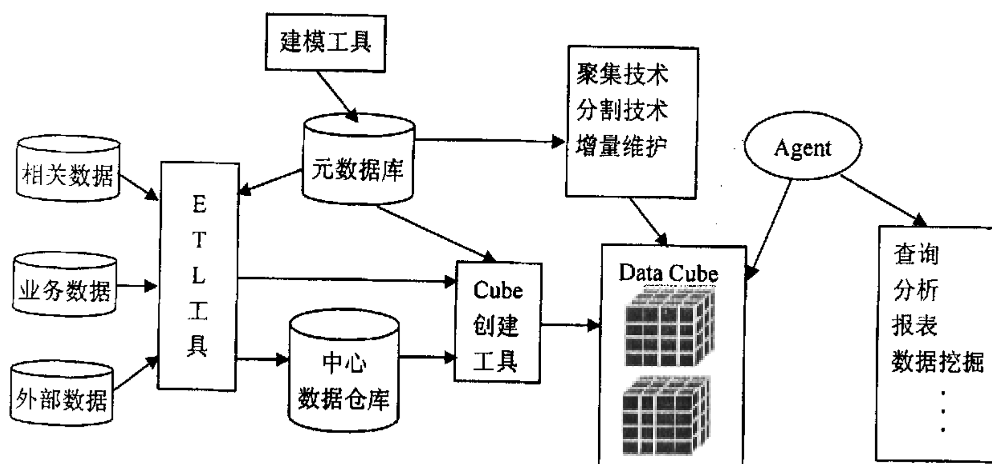


图 1-2　多维数据仓库原型系统的体系结构

在图 1-2 的原型系统中，数据仓库设计人员可以通过 SEUDW 的建模工具进行数据仓库模式设计；数据仓库实现人员通过 SEUDW 的 ETL 工具来设计和实现 ETL 过程，将企业内部业务数据和外部数据等数据源中的数据加载到企业级中心数据仓库中；通过 OLAP 系统中的 Cube 创建工具对 OLAP 立方体的数据模型进行设计及其数据的加载和更新，通过 OLAP 系统中的聚集技术、分割技术、增量维护等关键技术来对 Data Cube 中多维数据集进行预聚集处理，对实视图进行选择和有效分割，以及对实视图进行增量更新维护等。本文将主要对基于维层次编码的 ROLAP 聚集技术、维层次聚集 Cube、实视图的选择与分割及其增量更新维护等 OLAP 若干关键技术进行重点论述。

1.4.1 基于维层次编码的 ROLAP 聚集技术

充分利用维属性具有层次性以及维层次树中成员具有相同的前缀路径等特点，提出基于维层次编码的一种新型预分组聚集算法 DHEPGA（pre-grouping aggregation based on the dimension hierarchical encoding)。在标准的星形模式或雪花模式的维表和事实表中添加相应的维层次编码，即在每个维表中添加一个属性，用于存储根据其维层次树生成的各维成员的维层次编码，在事实表中为每一个维添加一个存储该维外键的维层次编码。通过这些比维表外关键字小得多且具有语义性质的维层次编码来代替事实表中的维表外键，可以直接在内存中对事实表的数据进行检索，快速检索出与检索关键字相匹配的维层次编码，来求得所有维层次属性的查询范围，提高了检索速度。大大减少和简化了事实表与维表之间的多表连接，减少了 I/O 开销，提高了 OLAP 查询效率。

1.4.2 维层次聚集 Cube

在数据仓库系统中设计高效的聚集 Cube 及其增量更新是建立高性能多维数据仓库系统的关键，尤其是对通过预聚集处理的聚集 Cube 进行增量更新，以往的 Cube 更新效率不太高，且增加了更新费用。本文提出利用 Cube 中的维层次(dimension hierarchy)聚集技术来创建高性能的维层次聚集 Cube（dimension hierarchy aggregate cube，简称 DHAC)。在用户提交 MOLAP 查询 Q 时，根据查询 Q 的查询空间 MBB_q 与聚集 Cube 中各结点的 MBB_c 进行比较，计算 Q 的查询结果，从而提高了 MOLAP 查询效率。在维层次聚集 Cube 中进行数据插入和删除等数据更新时，利用维层次聚集树中的维层次前缀，沿着维层次聚集树由下而上用更新前后的差值对更新单元所在结点的所有受更新影响的祖先结点进行增量更新，直到某一祖先结点不再更新为止，最差情况下更新到根结点。在插入新的维及其数据时，只需对维层次聚集 Cube 的范围 $MBB_c=\{c_1,c_2,\cdots c_d\}$ 中添加一个元素 c_{d+1} 即可，从而避免了以往 Cube 在插入新维时重构 Cube 所造成大量的时间消耗，实现了不需要重新构建聚集 Cube 就可对 DHAC 进行增量更新，提高了 Cube 的更新效率。

1.4.3 实视图选择与分割及增量更新维护

在数据仓库中，为了加快对大量数据的查询处理速度，通常将数据以实视图方式进行存储。OLAP 查询常常是非常复杂的，需要涉及大量的数据，因而使查询响应时间和维护费用成为数据仓库与 OLAP 中的重要问题。选择数据仓库视图进行有效分割和实体化，使 OLAP 查询通过访问较少的分割裂片以及较少的元组就可以完成，从而加快查询响应时间，削减维护费用。当基础数据发生变化时，这些实视图也需要定时的或按需进行更新维护，以确保查询的数据质量和数据的准确性，因而视图增量更新维护也成为数据仓库与 OLAP 中的另一个重要问题。

1.4.3.1 实视图的选择与分割

利用 OLAP 查询穷举其选择谓词以构造其最小项谓词，选择数据仓库立方体视图进行水平分割并实体化。将水平分割后的裂片定义以元数据的形式保存到元数据库中，相应的数据保存到分割的数据仓库或数据集市中。在决策者向 OLAP 服务器提交查询时，OLAP 服务器中的查询 Agent 根据用户提交的查询分解出其查询选择谓词，通过元数据库中的视图及其相应裂片的定义元数据从数据仓库或数据集市中提取相应的数据，来更好地回答用户提交的 OLAP 查询，从而提高 OLAP 查询响应时间和 DSS 的决策效率。

1.4.3.2 实视图的增量更新维护

通过视图表达式树的中间结果来创建辅助视图，并在数据仓库进行实体化。利用这些实体化的辅助视图和有效的视图增量维护算法来计算实视图和辅助视图的精确变化，实现数据仓库视图快速

的增量更新维护，从而缩短数据仓库更新维护时间，提高刷新速度和数据质量，来有效地改善 OLAP 查询效率。

1.5 本文的创新之处

本文有以下几个主要创新之处：

（1）提出了一种基于维层次编码的新型预分组聚集算法 DHEPGA。DHEPGA 充分利用维属性的层次性，通过维层次树对维各个层次的成员进行编码，来替代维表中关键字，实现对现有维关键字进行压缩，大大降低了存储空间。利用维层次前缀对事实表中的事实数据进行快速的分组聚集计算，将多表连接转化为对维层次编码范围的查询，大大减少和简化了事实表与维表之间的多表连接，减少了 I/O 开销，从而提高了 OLAP 查询效率。

（2）提出了利用 Cube 中的维层次聚集技术来创建高效的维层次聚集 Cube（DHAC）的方法。DHAC 利用 Cube 的维层次属性，对 Cube 中的数据单元进行预聚集处理，来生成层次聚集 Cube。在 DHAC 中进行数据插入和删除等数据更新时，通过维层次聚集树中的维层次前缀，由下向上对受到更新结点影响的所有祖先结点进行增量更新，提高了 Cube 更新效率；尤其是在插入新维数据时，可以对聚集 Cube 进行模式增量更新，从而避免了因 Cube 的重构所造成的时间开销。

（3）提出了一种数据仓库实视图选择与分割算法。利用 OLAP 查询中的选择谓词构造的最小项谓词，选择数据仓库视图进行有效的水平分割和实体化。通过元数据选择实体化的视图及其相应裂片以较少的元组来回答用户提交的 OLAP 查询，从而提高 OLAP 查询效率和 DSS 的决策效率。

（4）提出了一种数据仓库实视图增量更新维护的有效策略。基于视图表达式树，通过实体化的辅助视图，来实现实视图快速的增量更新维护，从而缩短数据仓库更新维护时间，有效地改善了 OLAP 查询的数据质量和效率。

第二章　基本概念

本章将简要地介绍数据仓库的基本概念及其基本数据模式，并对联机分析处理（OLAP）、位图索引与编码索引等基本概念进行阐述。

2.1　数据仓库

2.1.1 数据仓库基本概念

企业要想在日益激烈的竞争环境下生存和发展，建立一个决策支持系统（DSS）是非常必要的。但企业决策者目前已不仅仅满足于对数据进行查询，而是更希望能有效地对变化中的企业环境进行分析，不仅了解企业内部运行状况及市场的瞬息变化，而且还能掌握历史及发展趋势，得到更深层次的信息，以便更加有效地进行决策，这些需求可以通过数据仓库技术来实现[1,3,7]。数据仓库（Data Warehouse）是从企业内外部多个数据源采集有关数据，按照单一的模式进行存储，可以将与决策目标相关的各种应用系统集成在一起，为信息的集成处理和决策分析提供有效支持。

按照 W. H. Inmon 的说法，“数据仓库是一个面向主题的、集成的、时变的、非易失的数据集合，支持管理部门的决策过程”。这个定义阐明了数据仓库的四个主要特征：面向主题的、集成的、时变的、非易失的，将数据仓库与其他数据存储系统（如关系数据库系统、事务处理系统和文件系统）区别开来。

面向主题的（subject-oriented）：数据仓库是围绕某些决策主题的，如顾客、供应商、产品和销售等。数据仓库关注决策者的数据建模与分析，而不注重组织的日常操作和事务处理细节。因此，数据仓库排除对决策无用的数据，提供特定主题的简明视图。

集成的（integrated）：通常，构造数据仓库是将多个异种数据源，如关系数据库、一般文件和联机事务处理记录等集成在一起。使用数据清理和数据集成技术，确保命名约定、编码结构和属性度量等指标的一致性。

时变的（time-variant）：数据存储从历史的角度（例如过去 5-10 年）提供信息。数据仓库中的关键结构，隐式或显式地包含时间元素；数据仓库中的数据随着时间的变化定期的或按需进行更新。

非易失的（nonvolatile）：数据仓库中的数据源于操作环境下的应用数据，但在物理上与操作数据相分离，相对于事务数据是相当稳定的。

概言之，数据仓库是一种语义上一致的数据集合，它是决策支持数据模型的物理实现，并存放企业战略决策所需信息。数据仓库也常常被看作决策支持系统的一种体系结构，通过将异种数据源中的数据集成在一起，支持结构化的和专门的查询与分析，支持决策过程。

数据集市(datamart)是数据仓库的一个部门子集，它聚焦在部门范围的决策主题上。

2.1.2 数据仓库的数据模式

决策所需的数据总是按一定的模式来组织的。数据仓库的数据模式主要有以下几种：

1）星形模式

星形模式是将维信息和事实信息分别存储在维表和事实表中，以事实表为中心，通过事实表的外键与相关的维表关联起来。某销售数据仓库的星形模式如图 2-1 所示，它是由存储事实信息的销售事实表和存储维信息的时间维、产品维、地区维等维表组成。

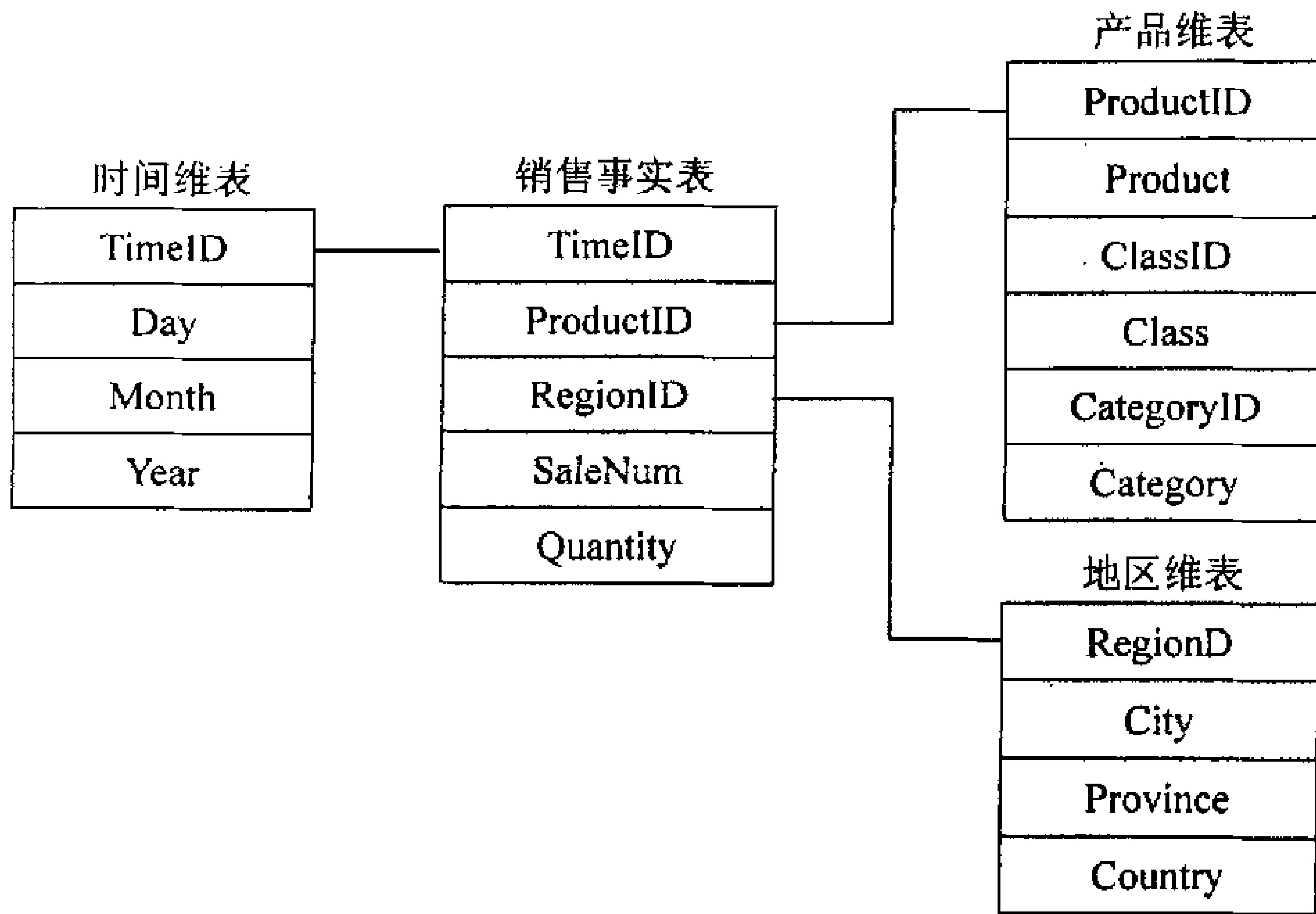


图 2-1　星形模式

维一般具有层次结构。图 2-1 中各个维的层次结构如下：

时间维的层次结构为：Day→Month→Year；

产品维的层次结构为：Product→Class→Category；

地区维的层次结构为：City→Province→Country。

2）雪花模式

若把维表按其层次结构表示，即为数据仓库的雪花模式，如图 2-2 所示。

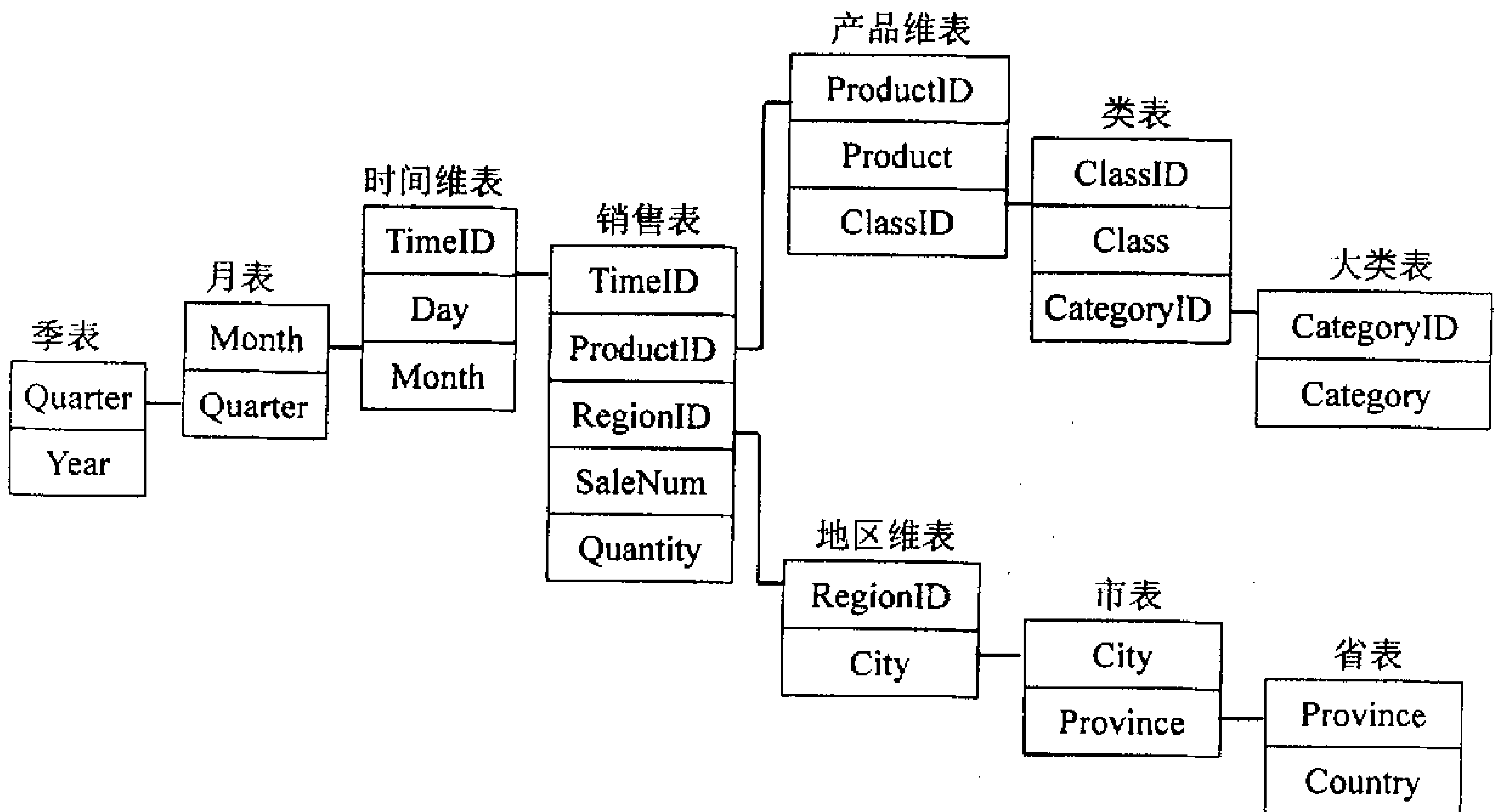


图 2-2　雪花模式

采用星形模式或雪花模式的主要优点是基于关系数据库存储数据，可以继续使用已经比较成熟的关系数据库的理论和实现技术；主要缺点是还须根据 OLAP 的特点增加一些功能，如采用新型索引技术[8]对在星形模式上的查询进行优化，但 OLAP 查询处理的性能仍然不太理想。

3）数据立方体

为了方便高效地进行多维数据分析，需要采用多维存储方式来进行多维数据的物理存储组织。这通常是以数据立方体（Data Cube）形式进行存储，如图 2-3 所示。

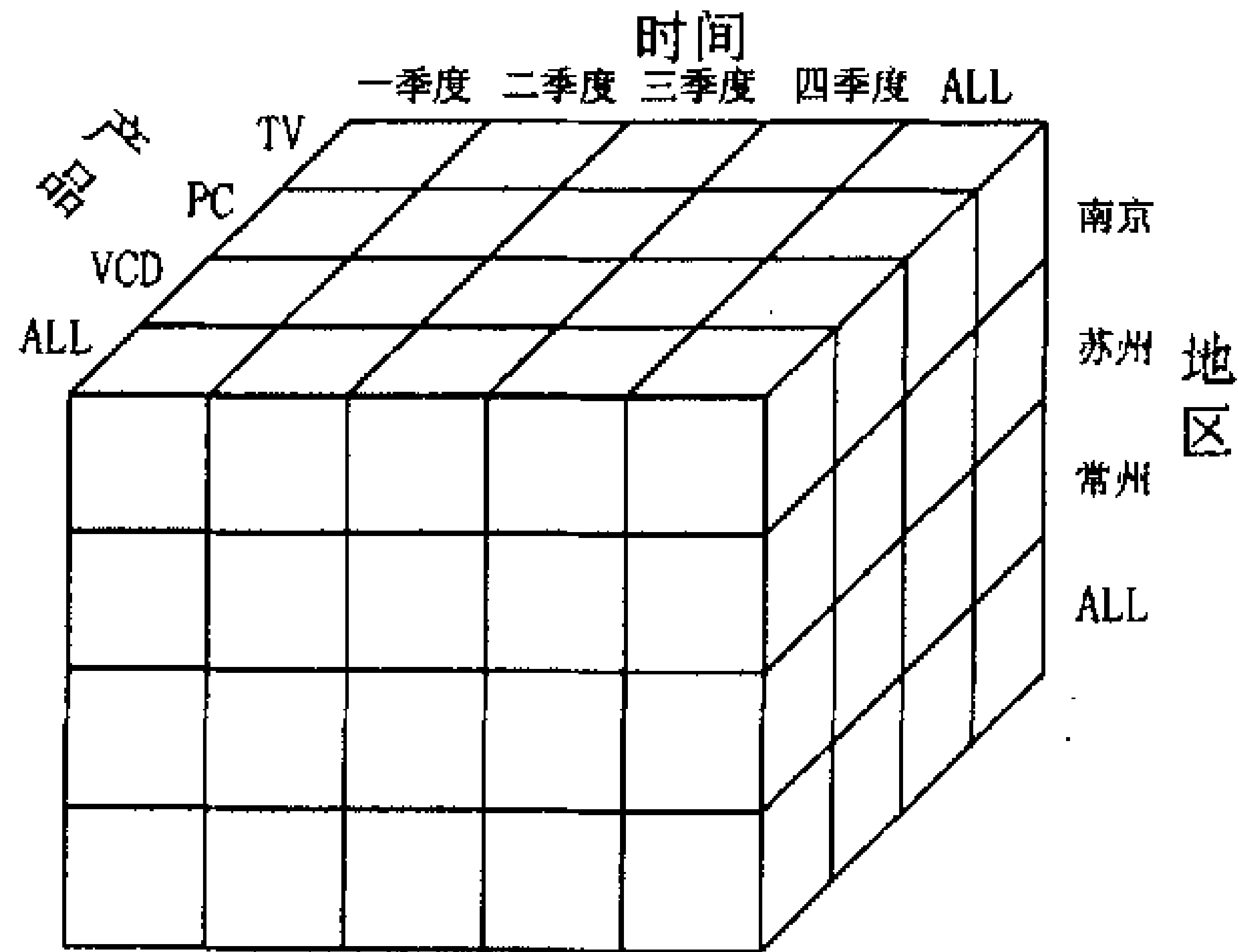


图 2-3　数据立方体（Data Cube）

在 Cube 中，多维数据的维属性无须存储，仅用作维索引，以确定多维数据集合中每个数据项在 Cube 中的位置，Cube 中仅存储度量属性值。对于多维数据集合 Cube（$D_1,D_2,\cdots,D_m,M_1,M_2,\cdots,M_n$），其中 $D_1,D_2,\cdots,D_m$ 分别维 1、维 2…、维 *m* 等相应维的取值集合，由这 *m* 个维的笛卡尔积来构成一个 *m* 维空间，$M_1,M_2,\cdots,M_n$ 表示 *n* 个事实属性，通常采用 *m* 维数组来存储这 *n* 个事实属性。采用多维数组直接存储多维数据时，能够在多维数组的基础上方便地进行各种 OLAP 操作，使得 OLAP 查询处理具有很快的响应速度。

2.2　联机分析处理（OLAP）

在数据仓库建立之后，即可利用 OLAP 复杂的查询能力、数据对比、数据抽取和报表来进行探测式数据分析了。之所以称其为探测式数据分析，是因为用户在选择相关数据后，通过切片、切块、上钻、下钻、旋转等操作，可以在不同的粒度上对数据进行分析尝试，得到不同形式的知识和结果。

联机分析处理（Online Analytical Processing，简称 OLAP）是一种软件技术，用于支持复杂的查询和分析操作，它使分析人员能够快速、一致、交互地从多种角度观察信息，以达到对数据进行更深入地理解。这些信息是从原始数据中转化过来的，它们以用户容易理解的方式反映企业的真实状况[1-7]。OLAP 主要有以下几个基本概念：

1）变量

变量是从现实系统中抽象出来的，用于描述数据的实际含义。比如，变量“身高”，其含义是指人的垂直高度。变量都有一定的取值范围，比如“体温”的正常变化范围是“36~37℃”。取值范围实际上是具体问题对变量的约束。

2）维

维是指与一事件相关的因素。比如客户销售产品这一事件中相关的因素有客户、时间、地区、产品等。

3）维的层次

维的层次是指一个因素的粒度。比如时间维可以分为“日”、“月”、“年”等层次。维的层次表

示人们观察数据的详细程度，维层次的确定需要具体问题具体分析，不同分析应用对数据详细程度的要求是不同的。

4）维成员

维的一个取值称为该维的一个维成员。由于维具有层次性，因此当维具有多个层次时，维成员则由各个层次的所有取值组合而成。比如，地区维由国家、省、市 3 个层次构成，“中国江苏省南京市”是地区维的一个维成员。

5）事实

事实是事件的某种度量，表示为不同维度在某一取值时的交叉点。如研究销售这个事件，那么在某组特定维（如时间、地区、产品）值时的销售量、销售额或销售成本都是事实。

6）多维数组

一个多维数组可以表示为：（维 1,维 2,⋯, 维 m,变量）。例如产品销售数据是按时间、地区和产品组织起来的三维立方体，加上变量销售额，就组成了一个多维数组（时间,地区,产品,销售额）。

7）数据单元

多维数组的取值称为数据单元。当多维数组的各个维都选中一个维成员，这些维成员的组合就唯一确定了一个变量的值。数据单元可以表示为（维 1 成员,维 2 成员,⋯, 维 m 成员,变量的值）。

OLAP 的主要功能是管理决策所需要的总结数据，而总结数据一般都以实视图的形式出现在数据仓库中[1]。实视图也是个表，可以用 RDBMS 管理，再根据 OLAP 的特点增加一些功能，对 SQL 做少许扩充，用这种方式实现的 OLAP 称为关系 OLAP（relational OLAP，简称为 ROLAP）。实视图又可组成立方体或超立方体结构，用这种方式实现的 OLAP 称为多维 OLAP（multidimensional OLAP，简称为 MOLAP）。ROLAP 是基于关系数据库存储方式建立的 OLAP，MOLAP 是基于多维数据库存储方式建立的 OLAP。ROLAP 在大数据量的存储上有绝对的优势，因此拥有海量数据的系统可以选择 ROLAP。MOLAP 在响应速度、预聚集和多维计算方面具有优势，中小型系统可以考虑采用 MOLAP。一般在企业级中心数据仓库采用 ROLAP，在部门级的数据集市中采用 MOLAP。

在实际的 MOLAP 系统中，通常事先对细节数据进行各种综合程度的预运算，并将运算结果保存到聚集 Cube 中。当用户需要查看各个综合程度的数据时，系统就可以直接读取已经计算好的结果展现给用户，这就是 OLAP 著名的“空间换时间”技术。由于磁盘空间和增量更新费用的限制，通常只对细节数据按部分综合程度进行预聚集处理。分析设计人员常常根据用户最常使用的功能将最常查看的数据有针对性地进行预聚集处理；而对于不常用的 OLAP 请求，一般不事先进行预聚集计算，只是在请求提交后再临时进行聚集计算。

2.3 编码索引

2.3.1 编码方法

在数据仓库中为了便于对数据进行快速查询，通常对维属性进行编码，编码方法主要有整数编码和二进制制编码两大类型[9,10,101]。

1）整数编码

整数编码主要是将给定的字符串集合 $S=\{Str_1,\cdots, Str_n\}$，构造一个映射 $f: S\rightarrow N$，使得 $f(Str_i)=i$，其中，$N=\{1,2,\cdots,n\}$。并且，若存在 $Str_i\leqslant Str_j$，则有 $f(Str_i)\leqslant f(Str_j)$成立。

在 MOLAP 中，可以采用多维数组存储结构来存储多维数据，通常采用整数编码方式将维的属性值（主要是维表中的各关键字值）映射成多维数组的下标值或下标的范围。

例 2.1 表 2.1 为地区维表关键字 RegionID 及其通过整数编码映射成多维数组下标值的映射表。

表 2.1　地区维表及其映射表

（a）地区维表

RegionID	Country	Province	City
No.1	中国	江苏	南京
No.2	中国	江苏	苏州
No.3	中国	浙江	杭州
No.4	中国	浙江	宁波
No.5	中国	安徽	合肥
No.6	中国	湖北	武汉

（b）地区维表映射表

RegionID	MapInt
No.1	0
No.2	1
No.3	2
No.4	3
No.5	4
No.6	5

2）二进制编码

二进制编码主要用一组 {0,1} 位串来对字符串进行编码。与整数编码方式相比，具有较强的数据压缩比。

定义 2.1（二进制编码）．设 S 是一个字符串集合 $S=\{Str_1,\cdots,\ Str_n\}$，且 $N=|S|$，则 S 中任意字符串的二进制编码是一个$\lceil \log_2 N \rceil$位的二进制数，即 S 的二进制编码 S^B 为 $\{e_1,\cdots,\ e_n\}$。其中，若存在 $Str_i \leq Str_j$，则有 $e_i \leq e_j$ 成立。

2.3.2 位图索引

位图索引[1,10]（Bitmap index）是一种新兴的索引结构，它也是一种二进制编码方法。在某些场合，它可以显著地提高性能和节省存储空间，在数据仓库中更显出其优势，并得到广泛的应用。

1）简单位图索引

我们常用的位图索引是用一组 {0,1} 位串来表示数据表中的某一列属性值，其 {0,1} 位串的位数为该列属性值不同取值的个数，它是一种简单位图索引。

定义 2.2（简单位图索引）．假设一个表 $DT=\{t_1,\ \cdots,t_j,\cdots,t_n\}$，$t_j$ 为表 DT 中的一个元组（j=1,2, $\cdots$,n），A 为表 DT 中的一个属性，其值域为$\{a_1,\ \cdots a_i,\cdots,a_m\}$，则在表 DT 的属性 A 的简单位图索引 B^A 是由位图向量集 $\{B_1,\ \cdots B_i,\cdots,B_m\}$ 组成，位图向量满足下列条件：

$\forall B_i$（i=1,2, $\cdots$,m），t_j（j=1,2, $\cdots$,n），如果 $t_j.A=a_i$，则 $B_i[j]=1$，否则 $B_i[j]=0$；

其中：$B_i[j]$表示位图向量 B_i 的第 j 位。

例 2.2 表 2.2 为表 2.1 中地区维表的 City 和 Province 两个维层次属性的简单位图索引。

表 2.2　简单位图索引

（a）Province 属性列

$B_{江苏}$	$B_{浙江}$	$B_{安徽}$	$B_{湖北}$
1	0	0	0
1	0	0	0
0	1	0	0
0	1	0	0
0	0	1	0
0	0	0	1

（b）City 属性列

$B_{南京}$	$B_{苏州}$	$B_{杭州}$	$B_{宁波}$	$B_{合肥}$	$B_{武汉}$
1	0	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0
0	0	0	0	0	1

对于简单位图索引，其 bit 的位数通常是该属性列不同属性值的个数 m，则该属性的简单位图索引共需要$|DT|*|A|/8=n*m/8$ 个字节存储空间（其中，n 为表 DT 中的元组个数，m 为该属性列的不同属性值的个数）。这种简单位图索引适用于低基数（low-cardinality）属性，对于高基数（low-cardinality）属性，在存储、加载、搜索、维护等方面将需要大量的时间和空间。

在简单位图索引中，位图向量的稀疏度为 $\frac{m-1}{m}\times 100\%$。随着 m 值的增大，空间利用率大大降低，对查询效率也有很大地影响。

2）编码位图索引

为了克服简单位图索引的这些缺点，可以采用具有较高压缩性的编码位图索引来对高基数的维属性进行编码索引。

定义 2.3（编码位图索引）. 假设一个表 $DT=\{t_1, \cdots, t_j, \cdots, t_n\}$，$t_j$ 为表 DT 中的一个元组（$j=1,2, \cdots, n$），A 为表 DT 中的一个属性，其值域为$\{a_1, \cdots a_i, \cdots, a_m\}$，则在表 DT 的属性 A 的编码位图索引 B^A 是由位图向量集 $\{B_{k-1}, \cdots B_0\}$、映射表（M^A: $A\rightarrow\{<b_{k-1}\cdots b_i\cdots b_0>|\ b_i\in\{0,1\}, i=0,\cdots, k-1\}$）（$k=\lceil\log_2 m\rceil$）等组成。位图向量满足下列条件：$\forall$ B_i（$i=1,2, \cdots, m$），t_j（$j=1,2, \cdots, n$），如果 $M^A(t_j.A)[i]=1$，则 $B_i[j]=1$，否则 $B_i[j]=0$。

其中：$B_i[j]$表示位图向量 B_i 的第 j 位，$M^A(t_j.A)[i]$表示 $M^A(t_j.A)$ 的第 i 位。

例 2.3 例 2.2 中的 City、Province 属性列上的编码位图索引如表 2.3 所示。

表 2.3 编码位图索引

（a）地区维表 Province 属性列

位图向量

B_1	B_0
0	0
0	0
0	1
0	1
1	0
1	1

映射表

Province	BitCode
江苏	00
浙江	01
安徽	10
湖北	11

（b）地区维表 City 属性列

位图向量

B_2	B_1	B_0
0	0	0
0	0	1
0	1	1
1	0	0
1	0	1
1	1	0

映射表

City	BitCode
南京	000
苏州	001
杭州	011
宁波	100
合肥	101
武汉	110

对于上例中有 4 个不同属性值的地区维表 Province 属性列只需要$\lceil log_2 4\rceil=2$ 个 bit 位，对于有 6 个不同属性值的地区维表 City 属性列只需要$\lceil log_2 6\rceil=3$ 个 bit 位，这比在简单位图索引中相对应的 6 个 bit 位和 4 个 bit 位要少。在编码位图索引中的编码位数只需要$\lceil log_2 m\rceil$个 bit 位，这比简单位图索引中需要 m 个 bit 位要少得多（其中，m 为该属性列的不同属性值的个数）。

第三章 基于维层次编码的 ROLAP 聚集技术

在 ROLAP 中，往往需在海量数据上进行即席的复杂分组聚集查询，在其 SQL 语句中通常包含多表连接和分组聚集操作，因而减少多表连接和压缩关键字，以及对查询数据进行有效地分组聚集操作，成为 ROLAP 查询处理的关键问题。

本章首先简述维层次编码基本理论，给出带有维层次编码的数据仓库多维模式；然后提出一种基于维层次编码的新型预分组聚集算法（DHEPGA 算法）；最后对算法性能进行分析和论证。

3.1 概述

OLAP 是用来对数据仓库中的数据进行联机分析的一种新技术[4,5,7]，其主要功能是管理决策所需要的总结数据，满足用户的即席（ad hoc）查询，及时向用户提供分析数据，来辅助决策，从而对查询响应速度提出了更高的要求。目前主要通过 MOLAP 和 ROLAP 两种方式来实现 OLAP 查询。在 ROLAP 中，主要采用星形模式和雪花型模式来存储多维数据，将其多维结构划分为事实表和维表，事实表是用来存储事实的度量值和各维的外键，维表一般有多个，对于每一个维，通常用一个表来保存该维的元组数，即维的描述信息。以事实表为中心，通过维表外键连接若干维表，组成星形结构。通常维表中的记录要比事实表中的记录少得多，而且一般不随着时间而积累，相对稳定。雪花模式则是将维表按其层次结构分成多个子表来保存维信息，这样可以节省存储空间，但是在访问维表时，则需要更多的维表与子表进行多表连接操作。

我们考虑充分利用维属性具有层次性及其维层次树中成员具有相同的前缀路径等特点，提出基于维层次编码的一种新型预分组聚集算法 DHEPGA。DHEPGA 算法充分利用维属性具有层次性特点，通过维层次树创建的维层次编码来代替维表中原关键字，实现维关键字的压缩；通过这种比维表外关键字小得多的维层次编码来代替事实表中的维表外键，快速检索出与检索关键字相匹配的维层次编码，来求得所有维层次属性的查询范围，提高了检索速度。利用维层次编码前缀对事实表中的数据进行快速预分组和层次聚集操作，可以提高事实表中的记录检索速度，大大减少和简化了事实表与维表之间的多表连接，提高了 OLAP 查询效率。

3.2 维层次编码

定义 3.1(维层次编码的星形模式). 数据仓库中带有维层次编码的星形模式可以用 $DW=(DT,FT)$来表示。其中 $DT=\{D_1, \cdots, D_m\}$是维表的集合，通常 $m\geq3$，$FT=\{D_1.fk, \cdots, D_m.fk, M_1, \cdots, M_n, B^{D_1}, \cdots, B^{D_m}\}$ 为事实表，$n\geq1$。

其中：$D_i=(Key, DH_i, DF_i, B^{D_i})$，$Key_i$ 表示维 D_i 的关键字属性集；DH_i 表示维 D_i 的所有层次的集合，它是一个有序集，即 $DH_i=\{(L_1^i, L_2^i, \cdots, L_h^i), \preceq\}$，$L_i^i$ 是维的层次，$\preceq$ 是 DH_i 中维层次之间的依赖关系或称偏序关系；DF_i 表示维 D_i 的所有非关键字属性集，即描述属性；B^{D_i} 为维 D_i 各成员的维层次编码；$D_i.fk$ 为维 D_i 的外键，M_i 为度量属性。$\forall L_i^i$，$L_j^i \in DH_i$ 且 $L_i^i \preceq L_j^i$，则 L_i^i 层次高于

层 L_j^i。DH_i 中层次的高低是反映数据综合程度的高低，层次越高表示数据的综合程度越高。它也是作为数据仓库中度量数据进行分组和聚集操作的依据，层次高的度量数据可以通过层次低的度量数据进行聚集计算得到，高层次可以作为低层次度量数据的分组依据。

带有维层次编码的星形模式是在标准的星形模式的维表和事实表中添加相应的维层次编码，进行转换而成，即在每个维表中添加一个属性，用于存储根据其维层次树生成的各个维成员的维层次编码，在事实表中为每一个维添加一个存储该维外键的维层次编码。通常，为节省存储空间，可以将维表外键从事实表中删去。

例 3.1 某企业销售数据仓库中带有维层次编码的星形模式如图 3-1 所示，在时间维表 DimTime、产品维表 DimProduct、地区维表 DimRegion 中分别添加其各个维成员的维层次编码 B^{TimeID}、$B^{ProductID}$、$B^{RegionID}$，在销售事实表 Sales 中为这三个维表添加其外键的维层次编码 B^{TimeID}、$B^{ProductID}$、$B^{RegionID}$。

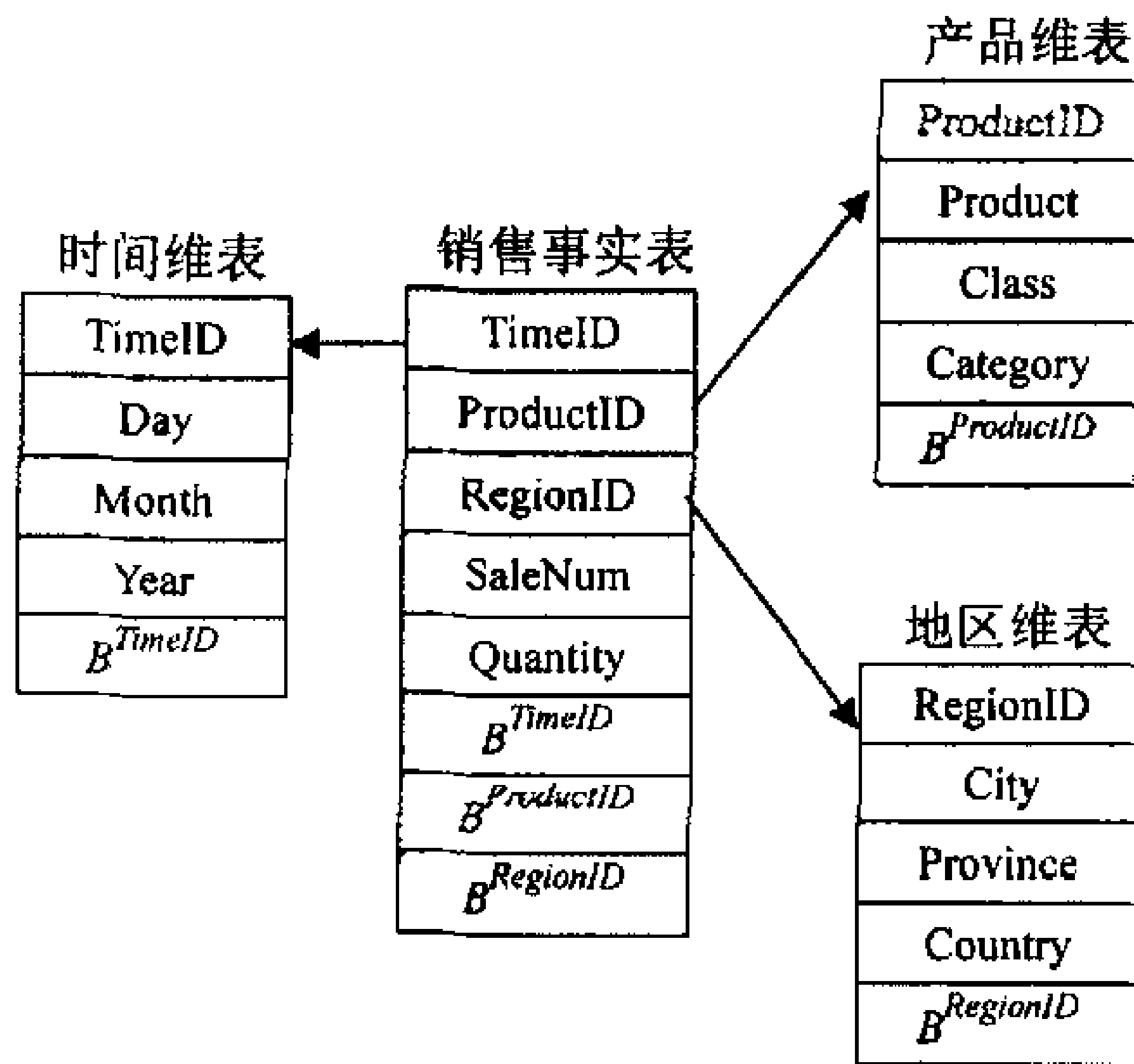


图 3-1 带有维层次编码的星形模式

定义 3.2（多维层次编码的雪花模式）. 在数据仓库雪花模式的各个维表及其子表中添加相应维成员的维层次编码属性，在事实表中为每一个维添加存储该维外键的维层次编码及所有维的维层次编码组合而成的多维层次编码。

例 3.2 某企业销售数据仓库中带有多维层次编码的雪花模式如图 3-2 所示。

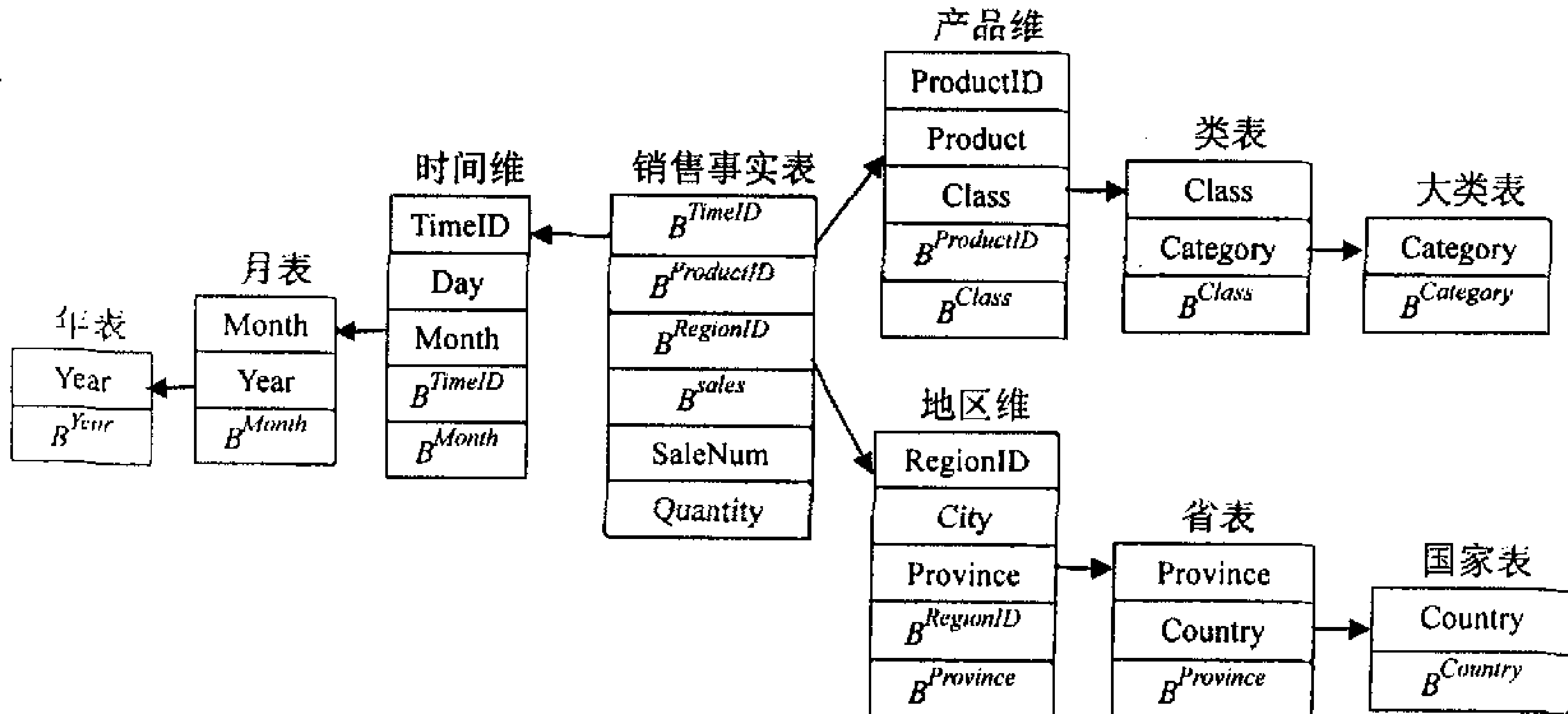


图 3-2 带有多维层次编码的雪花模式

其中，时间维表 DimTime、产品维表 DimProduct、地区维表 DimRegion 中分别添加其各维成员的维层次编码 B^{TimeID}、$B^{ProductID}$、$B^{RegionID}$，为年表、月表等各子维表建立其维层次编码；在销售事实表 Sales 中为这三个维表添加替代其维属性外键的维层次编码 B^{TimeID}、$B^{ProductID}$、$B^{RegionID}$，以及这三个维的维层次编码混合而成的多维编码 B^{sales}。事实表与维表之间通过维层次编码进行连接，来替代维表外键进行连接。

定义 3.3(维属性范围). 维属性范围是维 D_i在不同维层次 $\{L_1^i, L_2^i, \cdots, L_h^i\}$ 上的所有不同属性值的集合，用 $dom(D_i) = \{dom(L_1^i), \cdots, dom(L_h^i)\}$ 来表示。$dom(L_j^i)$ 表示维 D_i的第 j 层次不同取值的集合。

定义 3.4(维层次树). 维层次树可以用 $DTree=(V, E)$ 来表示，表示维中各个维成员及其层次关系。其中 $V = dom(D_i) = \cup_{j=1}^{h} dom(L_j^i)$ ($i=\{1, \cdots, k\}$) 是维中各个层次的所有成员的有序集；$E \subset dom(D_i) \times dom(D_j)$ 是各个维成员之间的层次关系，$\forall$ 维成员 $d_k^i \in dom(L_i^i)$、$d_l^j \in dom(L_j^i)$ 且 $L_i^i \preceq L_j^i$、$(d_k^i, d_l^j) \in E$，表示维成员 d_k^i 与 d_l^j 有层次依赖关系，即在维层次树中有边连接。

在传统的简单位图索引中，位图向量的稀疏度为 $\frac{m-1}{m} \times 100\%$ (m 为该属性列的不同属性值的个数)。随着 m 值的增大，空间利用率大大降低，对查询效率也有很大地影响。这种简单的位图索引适用于低基数（low-cardinality）属性，对于高基数（low-cardinality）属性，在存储、加载、搜索、维护等方面将需要大量的时间和空间。因此，我们考虑充分利用维属性具有层次特性，利用混合索引（B-Tree 和 Bit Code）技术来为每一个维的维层次属性进行二进制编码，利用维层次树生成维中各个成员的维层次编码，来代替维表中关键字，实现维关键字的压缩，大大降低存储空间。

定义 3.5(维层次属性二进制编码). 假设 L_j^i 为维表 D_i 中的第 j 层次属性，其值域为 $dom(L_j^i)=\{d_1^i, \cdots d_j^i, \cdots, d_m^i\}$，则在维表 D_i的维层次 L_j^i 各属性成员编码为 $B^{L_j^i}: dom(L_j^i) \rightarrow \{<b_{k-1} \cdots b_i \cdots b_0> | b_i \in \{0,1\}, i=0, \cdots, k-1\}$。

其中，k 为维 D_i 的维层次 L_j^i 中成员二进制编码的位数，通常取值为 $k = Bit_{L_j^i} = \lceil log_2 m \rceil$ ($m = max(|L_j^i|)$ 为 L_j^i 层中不同成员的最大个数)。

性质 3.1 维 D_i中维层次 L_j^i 的层次编码位数 $Bit_{L_j^i}$ 为 $\lceil log_2 m \rceil$ ($m=max(|L_j^i|)$)，维 D_i层次编码的总位数 Bit_{D_i} 为 $\sum_{j=1}^{h} Bit_{L_j^i}$。其中 $max(|L_j^i|)$ 为 L_j^i 层的最大不同取值个数。对于最高层次 L_1^i，其最大取值个数就是该层中有多少个不同的取值；对于其他较低层次 L_{j+1}^i ($j=1, \cdots, h$)，其最大取值个数是该层对应于相同的较高层次 L_j^i 的不同取值个数中的最大值。

定义 3.6(维层次编码). D_i 维各成员的维层次编码是由 D_i 维中所有维层次属性（$L_1^i, L_2^i, \cdots, L_h^i$）的二进制编码按式 3.1 根据维层次由高到低依次进行组合而成的混合编码，即:

$$B^{D_i} = (\cdots((B^{L_1^i} << Bit_{L_2^i} \mid B^{L_2^i}) << Bit_{L_3^i} \mid B^{L_3^i}) \cdots) << Bit_{L_h^i} \mid B^{L_h^i} \quad (3.1)$$

例 3.3 某销售数据仓库中的时间维 DimTime 及其各个维层次编码见表 3.1，其维层次树如图 3-3 所示。在建立星形或雪花模式的数据仓库时，针对时间维 DimTime 建立了 Year、Month、Day 等层次编码表及其维层次树，针对产品维 DimProduct 建立了 Category、Class、Product 等层次编码表及其维层次树，针对地区维 DimRegion 建立了 City、Province、Country 等层次编码表及其维层次树。

表 3.1 维层次编码表

Year 层次编码表

Year	B^{Year}
1998	00
1999	01
2000	10
…	…

Month 层次编码表

Month	B^{Month}
1	0000
2	0001
…	…
12	1001

Day 层次编码表

Day	B^{Day}
1	00000
2	00001
…	…
31	11110

时间维 DimTime 多维层次编码表

TimeID	Time	B^{TimeID}
1	98.1.1	00000000000
…	…	…
366	99.1.1	01000000000
…	…	…

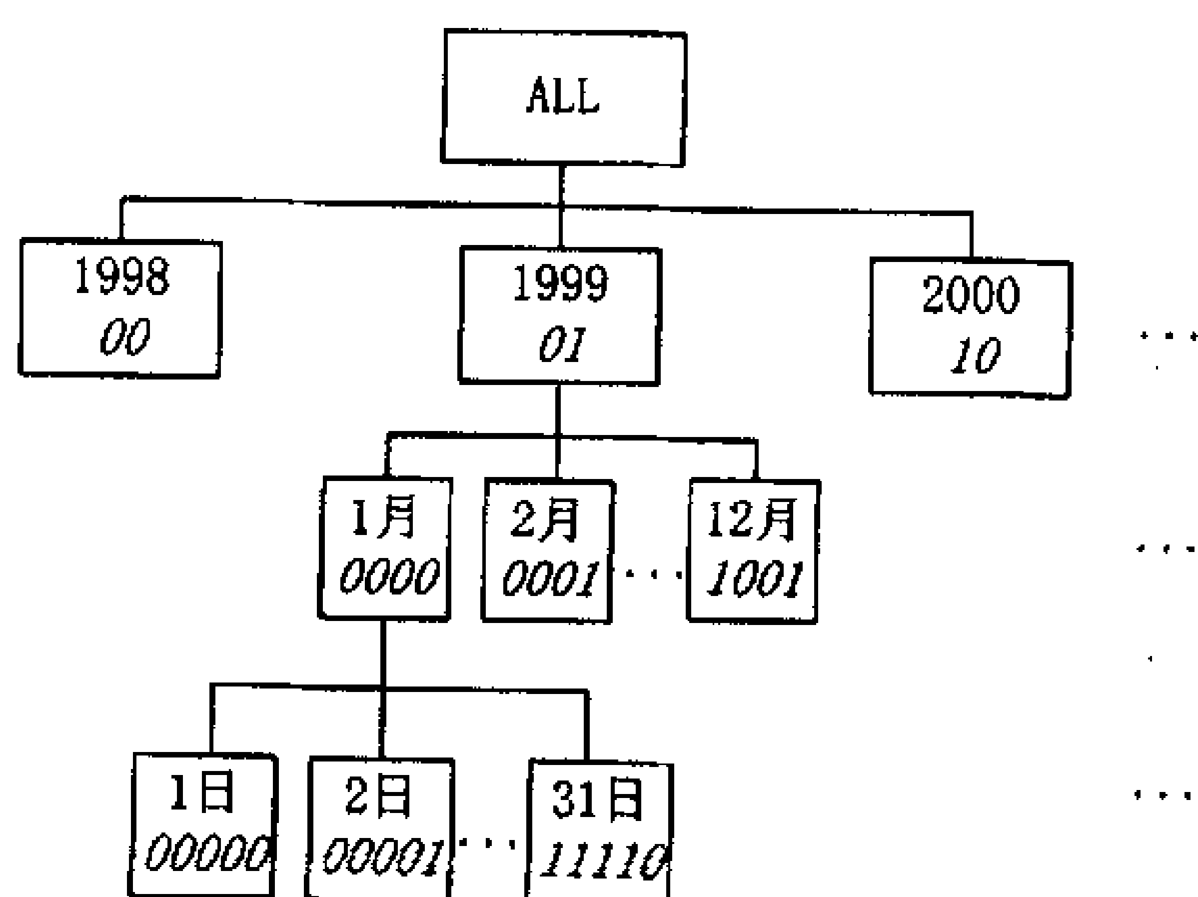


图 3-3 维层次树

在图 3-3 中，时间维 DimTime 共有三个不同层次{*Year,Month,Day*}，对于最高层次“年份(*Year*)”来说，不同成员的最大个数就是维 DimTime 中年份的不同取值个数，其最大个数为 3，其编码位数 Bit_{Year} 为 $\lceil log_2 3 \rceil$=2 *bit*；同理对于层次“月份(*Month*)”来说，由于一年最多有 12 个月，其最大个数为 12，其编码位数 Bit_{Month} 为 4 *bit*；对于层次“日期（*Day*)”来说，其最大个数为 31，其编码位数 Bit_{Day} 为 5 *bit*；时间维 DimTime 的编码总位数为 $Bit_{DimTime} = Bit_{Year} + Bit_{Month} + Bit_{Day}$= 11 *bit*。相对于关键字 *TimeID* 有较大的数据压缩。

维 DimTime 中维成员“1999 年 1 月 2 日”的维层次编码为 01000000001，它是由 $B^{Year=1999}B^{Month=1}B^{Day=2}$ 进行组合而成，即 $B^{1999年1月2日} = (B^{Year=1999} << Bit_{Month} \mid B^{Month=1}) << Bit_{Day} \mid B^{Day=2}$ = (01<<4 | 0000) <<5 | 00001=01000000001。

定义 3.7(多维层次编码). 事实表中各个记录的多维层次编码 B^{Fact} 是由组成该记录中各个维 D_i 中相应维成员的维层次编码按一定的顺序组合而成的混合编码，即按式 3.2 进行位移和位或运算而成的混合编码。

$$B^{Fact}=(\cdots((B^{D_1} << Bit_{D_1} \mid B^{D_2}) << Bit_{D_2} \mid B^{D_3}) \cdots) << Bit_{D_m} \mid B^{D_m} \quad (3.2)$$

对于多维层次编码 B^{Fact} 的各个维的组合顺序可以依据用户查询中的各个维出现的频率来确定的优先级（或用户自己根据查询的特点来确定的各个维的优先级），用函数 $prior(D_i)$来表示维的优先级，则 $prior(D_1) < prior(D_2) < \cdots < prior(D_m)$。

例 3.4 对于上述的销售数据仓库，假设用户在产品维 DimProduct 访问频率最高，在地区维 DimRegion 访问频率次之，在时间维 DimTime 访问频率最低，则 $prior$(DimTime) < $prior$(DimRegion) < $prior$(DimProduct)。因而销售事实表 Sales 中的多维层次编码 B^{Sales} 可以按 $B^{TimeID}B^{RegionID}B^{ProductID}$ 顺序进行组合而成。

以上是对维层次属性采用二进制编码来生成各个维成员的维层次编码。同理也可以根据需要对各个维的维层次属性采用整数编码，只要将该维的各个维层次属性（$L_1^i, L_2^i, \cdots, L_h^i$）的整数编码按层次由高到低依次进行合并，即可生成其各个维成员的维层次编码，可以采用字符串操作来对其进行分组聚集等 OLAP 查询常用操作。

3.3 DHEPGA 算法

在星形或雪花结构组成的数据仓库中，OLAP 查询一般都涉及多表连接和分组聚集操作，其标准 SQL 语句如下：

SELECT *S*, *Agg* FROM *FT*,*DT*
WHERE *JC* AND *LP* AND *FP*
GROUP BY GA_h, GA_f, GA_m
HAVING *HP*
ORDER BY *OP*

其中：*S* 为维表选择属性和事实表中度量属性；
Agg 为计算的聚集值；
FT 为事实表；
DT 为维表；
JC 为事实表和维表的自然连接条件；
LP 为各维表本地谓词的连接表达式，即 $LocPredicate_1(D_1) \wedge LocPredicate_2(D_2) \wedge \cdots \wedge LocPredicate_m(D_m)$；
FP 是事实中度量属性的约束条件；
GA_h 为各个维的层次分组属性集，即 $\{GA_{h1}, GA_{h2}, \cdots, GA_{hm}\}$；
GA_f 为各个维表分组描述属性集；
GA_m 为事实表分组度量属性集；
HP 为分组和聚集条件表达式；
OP 为排序属性。

利用维层次编码及其前缀特性，可以实现对事实表进行层次簇集存储和预分组聚集操作，通过物理簇集可以大大降低数据仓库中基于维层次的 OLAP 查询的 I/O 开销，从而提高 OLAP 查询效率。为了更加有效地进行分组聚集计算，需要事先通过数据仓库建模工具创建维层次编码。通过数据仓

库建模工具，在建立星形或雪花结构的数据仓库的同时，根据星形或雪花结构中各个维表及其维层次属性，创建维层次树及其各个层次成员的维层次编码，并保存到相应编码表中，在事实表中将这些维层次编码作为维表外键的代替键存储起来。在维表中数据更新时，可以对维层次树及其维层次编码进行增量更新。如果插入新记录，则根据上述定义 3.5、定义 3.6 和性质 3.1，来创建这插入新记录的维层次编码。当插入新记录后，其 $Bit_{L_j^i}$ 大于插入记录前的 $\lceil log_2 m \rceil$（$m=max(|L_j^i|)$）时，则在现有的编码 $B^{L_j^i}$ 前统一都加一个 bit 0，即 $B^{L_j^i.new}=0<<Bit_{L_j^i.old}|B^{L_j^i.old}$，接着按编码的次序对插入的新记录进行编码（其中 $B^{L_j^i.new}$、$B^{L_j^i.old}$ 分别为插入新记录前后的现有维成员的编码，$Bit_{L_j^i.old}$ 为插入新记录前的现有维成员的编码位数）。

例 3.5 在上例的 DimTime 时间维的维层次 *Year* 中增加成员 2001 年，则在维层次 *Year* 编码表中增加编码 11 表示其维层次编码 $B^{Year=2001}$。如果再增加成员 2002 年时，则插入后的维层次 *Year* 的成员个数为 5，其编码位数 Bit_{Year} 为 3 bit，大于插入前的 2 bit，则对插入新记录的 *Year* 层次 2002 年前的 1998 年、1999 年、2000 年、2001 年原编码 $B^{Year=1998}$=00、$B^{Year=1999}$=01、$B^{Year=2000}$=10、$B^{Year=2001}$=11，通过 $B^{Year.new}=0<<Bit_{Yaer.old}|B^{Yaer.old}$ 进行计算其新编码为 $B^{Year=1998}$=0<<2|00＝000、$B^{Year=1999}$=0<<2|01＝001、$B^{Year=2000}$=0<<2|10＝010、$B^{Year=2001}$=0<<2|11＝011，即在这四维成员的原编码前都统一添加了一个 bit 0。接着对插入新记录的 *Year* 层次成员 2002 年进行编码，即 $B^{Year=2002}$=100。

性质 3.2 层次 L_j^i 中的维成员 d_k^i 所在结点前缀路径 *DMPrefixpath*（*DTree*, d_k^i）是由该维成员 d_k^i 的所有祖先结点 *Ancestor*（d_k^i）的有序集，即 *DMPrefixpath*（*DTree*, d_k^i）= $\cup_{j=i}^{1}$ *DMPrefixpath*（*DTree*,*Parent*（d_k^j））={*Ancestor*（d_k^i）}。

性质 3.3 层次 L_j^i 中的维成员 d_k^i 的维层次编码前缀是该成员 d_k^i 所在结点前缀路径（即其所有祖先结点 *Ancestor*（d_k^i）成员）的维层次编码，分为直接前缀和间接前缀。直接前缀 *BPrefix*（$B^{d_k^i}$, L_{j-1}^i）是其父亲结点 *Parent*（d_k^i）的维层次编码 $B^{Parent(d_k^i)}$。间接前缀 *Bprefix*（$B^{d_k^i}$, L_{m-1}^i）（其中 m=1, …, j-1）是其所有祖先结点 *Ancestor*（d_k^i）的维层次编码（除其父亲结点外）。

定理 3.1 层次 L_j^i 中的维成员 d_k^i 的维层次编码前缀可以按式 3.3 对该成员维层次编码进行快速位移运算得到。

$$Bprefix(B^{d_k^i}, L_{m-1}^i) = B^{d_k^i} >> \sum_{l=m}^{j} (Bit_{L_l^i}) \text{（其中 } m=1 \cdots j \text{ 为编码前缀所在层次）} \quad (3.3)$$

证明：可用数学归纳法证得

当 $m=j$ 时，由定义 3.6 可得，层次为 L_j^i 的维成员 d_k^i 维层次编码为：

$$B^{d_k^i} = ((\cdots((B^{L_1^i} << Bit_{L_2^i} | B^{L_2^i}) << Bit_{L_3^i} | B^{L_3^i}) \cdots) << Bit_{L_{j-1}^i} | B^{L_{j-1}^i}) << Bit_{L_j^i} | B^{L_j^i} \quad (1)$$

层次为 L_j^i 的维成员 d_k^i 的父亲结点 *Parent*（d_k^i）的层次为 L_{j-1}^i，由定义 3.6 可得，其维层次编码为：

$$B^{Parent(d_k^i)} = (\cdots((B^{L_1^i} << Bit_{L_2^i} | B^{L_2^i}) << Bit_{L_3^i} | B^{L_3^i}) \cdots) << Bit_{L_{j-1}^i} | B^{L_{j-1}^i} \quad (2)$$

由式（1）和式（2）可得，层次为 L_j^i 的维成员 d_k^i 在层次 L_{j-1}^i 的编码前缀为：

$$Bprefix(B^{d_k^i}, L_{j-1}^i) = B^{Parent(d_k^i)} = B^{d_k^i} >> Bit_{L_j^i} \quad (3)$$

当 $m=j-1$ 时，按式（3）得：

$$Bprefix(B^{d_k^i}, L_{m-1}^i) = Bprefix(Bprefix(B^{d_k^i}, L_{j-1}^i), L_{j-2}^i) = BPrefix(B^{Parent(d_k^i)}, L_{j-2}^i)$$

$$=B^{Parent(d_k^i)} >> Bit_{L_{j-1}^i} = B^{d_k^i} >> Bit_{L_j^i} >> Bit_{L_{j-1}^i} = B^{d_k^i} >> (Bit_{L_j^i} + Bit_{L_{j-1}^i}) = B^{d_k^i} >> \sum_{l=j-1}^{j} (Bit_{L_l^i});$$

同理，由数学归纳法可以证得，层次 L_j^i 中的维成员 d_k^i 在任一维层次的编码前缀为：

$$Bprefix(B^{d_k^i}, L_{m-1}^i) = Bprefix(\cdots BPrefix(B^{Parent(d_k^i)}, L_{j-2}^i)\cdots) = B^{d_k^i} >> \sum_{l=m}^{j} (Bit_{L_l^i}) \quad (4)$$

将（3）和（4）合并得层次 L_i^i 中的维成员 d_k^i 的编码前缀的计算公式为

$Bprefix(B^{d_k^i}, L_{m-1}^i) = B^{d_k^i} >> \sum_{l=m}^{j} (Bit_{L_l^i})$（其中 $m=1 \cdots, j$ 为编码前缀所在层次）。

证毕。

例 3.6 在上述事例中，维层次 1999 年 1 月 2 日的前缀路径 *DMPrefixpath*（*DTimeTree*,1999 年 1 月 2 日）={1999 年 1 月,1999 年}，其编码直接前缀为 *BPrefix*（$B^{1999年1月2日}$,*Month*）= $B^{1999年1月}$= $B^{1999年1月2日}$>>Bit_{Day}=01000000001>>5=010000，在 *Year* 层的编码间接前缀为 *BPrefix*（$B^{1999年2月1日}$,*Year*）= $B^{1999年1月2日}$>>（Bit_{Day}+ Bit_{Month}）=01000000001>>（5+4）=01。

将定理 3.1 中的维层次属性编码及其编码位数用各个维成员编码及其编码位数来代替，按定理 3.1 可以类推出事实表中各个记录的多维层次编码前缀，见定理 3.2。

定理 3.2 事实表中各个记录的多维层次编码前缀可以按式 3.4 对该记录的多维层次编码 B^{Fact} 进行快速位移运算得到。

$$Bprefix(B^{Fact}, D_i) = B^{Fact} >> \sum_{l=j}^{i} (Bit_{D_l}) \quad \text{（其中 } j=1, \cdots, i \text{ 为维次序）} \quad (3.4)$$

证明：具体证明略。

为了加快查询记录的分组聚集计算速度，将查询 Q 所涉及的各个维的层次分组属性集 GA_h $\{GA_{h1}, GA_{h2}, \cdots, GA_{hm}\}$ 中各个分组属性 GA_{hi} 的层次编码 $B^{GA_{hi}}$ 按式 3.5 进行组合运算得到 GA_h 的混合编码 B^{GA_h}，作为对事实表中筛选的记录进行分组聚集计算的依据。

$$B^{GA_h} = (\cdots((B^{GA_{h1}} << Bit_{GA_{h2}} | B^{GA_{h2}}) << Bit_{GA_{h3}} | B^{GA_{h3}}) \cdots) << Bit_{GA_{hm}} | B^{GA_{hm}} \tag{3.5}$$

式中：B^{GA_h} 为维层次分组属性集 GA_h 的混合编码；

$B^{GA_{hi}}$、$Bit_{GA_{hi}}$ 分别为 GA_h 中所包含的各个维的层次分组属性 GA_{hi} 的维层次编码和编码位数（i=1, ⋯,m）。其中，$Bit_{GA_{hi}} = \sum_{l=j}^{i} (Bit\, L_l^i)$（$L_j^i$ 为分组属性 GA_{hi} 中层次最低即粒度最细的维层次，j 为其所在层次的层数）。

通过各维的层次分组属性 GA_{hi} 的位掩码 $BMask_{GA_{hi}}=2\uparrow Bit_{GA_{hi}}-1$（↑表示指数幂运算），对维层次分组属性集 GA_h 的混合编码 B^{GA_h} 按式 3.6 进行反算出各个维的层次分组属性 GA_{hi} 的维层次编码 $B^{GA_{hi}}$。

$$B^{GA_{hm}} = B^{GA_h} \,\&\, BMask_{GA_{hm}};$$

$$B^{GA_{hi}} = (\cdots(B^{GA_h} >> Bit_{GA_{hm}}) \cdots) >> Bit_{GA_{h(i+1)}} \,\&\, BMask_{GA_{hi}} \tag{3.6}$$

根据用户提交的 OLAP 查询 LP 中所涉及到的各个维的维层次编码的查询范围 B^{QD_i}，从事实表中筛选记录，并根据维层次分组属性集 GA_h 中的混合编码 B^{GA_h}，对筛选出的记录进行分组聚集运算。算法如下：

算法 3.1: DHEPGA 分组聚集算法

{/*输入：事实表 FT，维表 DT= {$D_1, \cdots, D_m$}；查询 Q；进行分组聚集计算的缓冲区 B；
  输出：OLAP 聚集查询结果*/
  首先对查询 Q 进行分析，得到各个维的本地谓词的连接表达式 LP 和维层次分组属性集 GA_h；
  For (i=1 to m)
    { 根据 LP 中所涉及到的各个维的本地谓词表式 $LocPredicate_i(D_i)$，求得这些涉及维的维层次编码 B^{D_i} 的查询范围 B^{QD_i}；
      根据 GA_h 中所涉及的各个维的维层次分组属性集，求得这些涉及的各个维的维层次分组属性编码 $B^{GA_{hi}}$；}

  $B^{GA_h} = B^{GA_{h1}}$；　/* 按式 3.5 来求得维层次分组属性集 GA_h 的混合编码 B^{GA_h} */
  For (i=1 to m)

{ $B^{GA_h} = B^{GA_h} <<Bit_{GA_{h(i+1)}}$；

$B^{GA_h} = B^{GA_h} | B^{GA_{h(i+1)}}$； }

While （事实表中仍有记录要访问）

{ 顺序扫描事实表 *FT*，从事实表选取其各个维层次编码 B^{D_i} 的前缀在查询范围 B^{QD_i} 内、并按 *FP* 条件进行过滤的各条记录;

根据每一个筛选出的记录，按各个维层次编码 B^{D_i} 的前缀为分组属性编码 $B^{GA_{hi}}$ 组合而成的混合编码 B^{GA_h} 簇集存储到缓冲区 B 中；

If （缓冲区 B 满）

对缓冲区 B 中各个簇集存储的记录按编码 B^{GA_h} 进行分组聚集计算，将运算结果 *Agg* 再按照 B^{GA_h} 簇集存储到缓冲区 B 中，将已参与运算过的记录从缓冲区中删去；

对缓冲区 B 中的各条记录按编码 B^{GA_h} 进行分组聚集计算得运算结果 *Agg*;

While（缓冲区 B 不为空）

{ 从缓冲区 B 选取记录，将选取记录的编码 B^{GA_h} 按式 3.6 反算出各个维的层次分组属性 GA_{hi} 的维层次编码 $B^{GA_{hi}}$；

将 $B^{GA_{h1}}$,…, $B^{GA_{hm}}$ 和聚集计算结果 *Agg* 作为按维层次分组属性进行分组聚集的结果写入外存中； }}

对外存中的按维层次属性进行聚集运算后的记录，按 GA_f、GA_m 等非维层次属性进行连接和聚集计算，再分别按 *HP*、*OP* 中等非维层次属性的分组和排序条件进行分组和排序，向用户提交的 OLAP 聚集查询结果;

}

为了提高事实表中记录的选取速度，对事实表中的记录也可以事先按维顺序及其维层次前缀进行簇集存储。可以对各个维的层次编码 B^{D_i} 按定义 3.7 计算出多维层次编码 B^{Fact}=（…（（$B^{D_1} <<Bit_{D_2} | B^{D_2}$）$<<Bit_{D_3} | B^{D_3}$）…）$<<Bit_{D_m} | B^{D_m}$，对事实中的记录进行簇集存放到外存中，将有助于按查询范围对事实表中的记录进行快速存取，进一步减少 I/O 的开销。

例 3.7 是对 DHEPGA 算法进行示例说明。

例 3. 7 假设用户提交查询 Q_1，来查询 1999 年南京市各产品大类的月销售额，其 SQL 语句为:

```
SELECT Month,City,Category , SUM(SaleNum)
FROM    DimTime, DimRegion, DimProduct ,Sales
WHERE DimTime.TimeID = Sales.TimeID
```

```
AND DimRegion.RegionID = Sales.RegionID
AND DimTime.ProductID = Sales.Product ID
AND DimTime.Year=1999 AND DimRegion.City ='南京'
GROUP BY Month ,City, Category
```

DHEPGA 算法的主要执行步骤如下：

（1）按 WHERE 子句中维表的连接谓词 DimTime.Year=1999 AND DimRegion.City ='南京'，分别从时间维 DimTime 和地区维 DimRegion 的维层次树计算并筛选出维层次编码为 B^{1999}=01 和 $B^{南京}$ 的查询范围。

（2）根据查询范围 B^{1999}=01 和 $B^{南京}$从事实表筛选出其维层次编码前缀 *BPrefix*（B^{TimeID}, *Year*）=01 和 *BPrefix*（$B^{RegionID}$, *City*）=$B^{南京}$的各记录，分别按其编码前缀 *BPrefix*（B^{TimeID}, *Month*）、*BPrefix*（$B^{RegionID}$, *City*）、*BPrefix*（$B^{ProductID}$, *Category*）为分组属性编码 B^{Month}、B^{City}、$B^{Category}$ 组合而成的混合编码 $B^{Month}B^{City}B^{Category}$ 簇集存储到缓冲区 B 中。

（3）对缓冲区 B 中的各记录按维层次属性 GA_h 的维层次编码 $B^{Month}B^{City}B^{Category}$ 进行分组聚集计算，即对具有相同的编码前缀 *BPrefix*（B^{TimeID}, *Month*）、*BPrefix*（$B^{RegionID}$, *City*）、*BPrefix*（$B^{ProductID}$, *Category*）的各记录进行聚集计算，将运算结果 *AGG* 再按照混合编码 $B^{Month}B^{City}B^{Category}$ 簇集存储到缓冲区 B 中，将已参与过聚集计算的记录从缓冲区 B 中删去。

（4）从缓冲区 B 选取记录，将选取记录的编码 $B^{Month}B^{City}B^{Category}$ 反算出各个维的层次分组属性 Month、City、Category 的维层次编码 B^{Month}、B^{City}、$B^{Category}$，将 B^{Month}、B^{City}、$B^{Category}$ 和聚集计算结果 *AGG* 作为按维层次分组属性进行分组聚集的结果写入外存中。

（5）向用户提交查询 Q_1 的结果。

因为查询 Q_1 的 SQL 语句中没有涉及到 GA_f、GA_m 等非维层次属性和 *HP*、*OP* 中非维层次属性的分组和排序条件，就可以直接将已按维层次属性进行聚集计算过的记录集提交给用户。

由上述的算法步骤和示例可以看出，DHEPGA 算法使用各个维的层次分组属性 GA_{hi} 的层次编码 $B^{GA_{hi}}$，进行位移和位或操作来组合运算得到 GA_h 的混合编码 B^{GA_h} 作为分组聚集计算依据，以及在聚集计算完成后将编码 B^{GA_h} 反算出各个维的层次分组属性 GA_{hi} 的维层次编码 $B^{GA_{hi}}$，虽然增加了一些额外操作，但是这些都是简单而快速的二进制操作，增加的额外开销并不是很大。另一方面，通过维层次编码及其前缀，可以将 OLAP 查询中大量的多表连接转换为在维上进行范围查询，大大减少和简化了事实表与维表之间的多表连接，利用维层次编码前缀和分组属性编码就可以直接对事实表中的记录进行分组聚集计算，将结果簇集存储到外存中，大大减少了 I/O 访问开销，从而提高了 OLAP 查询效率。

3.4 性能分析

在数据仓库中，我们常用的简单位图索引是用一组{0,1}位串来表示数据表中的某一列属性值，其{0,1}位串的位数为该列属性值的不同取值个数 *m*。随着该列维属性值不同取值个数 *m* 的增大，所需要的 bits 位数将也随着增加。因此，为了解决这些问题，利用维属性具有层次特性和位图编码（Bit Code）技术来为维属性进行二进制编码，生成各个维成员的维层次编码。在维层次编码中，一般只需要$\lceil log_2 m \rceil$个 bits（*m* 为不同属性值的成员个数），这比简单位图索引所需要的 *m* 个 bits 要少得多。

第1组实验测试是维层次编码与简单位图索引的压缩性能。测试数据为维属性值不同的元组个数为4~4096个，它们需要的编码位数对比关系如图3-4所示。

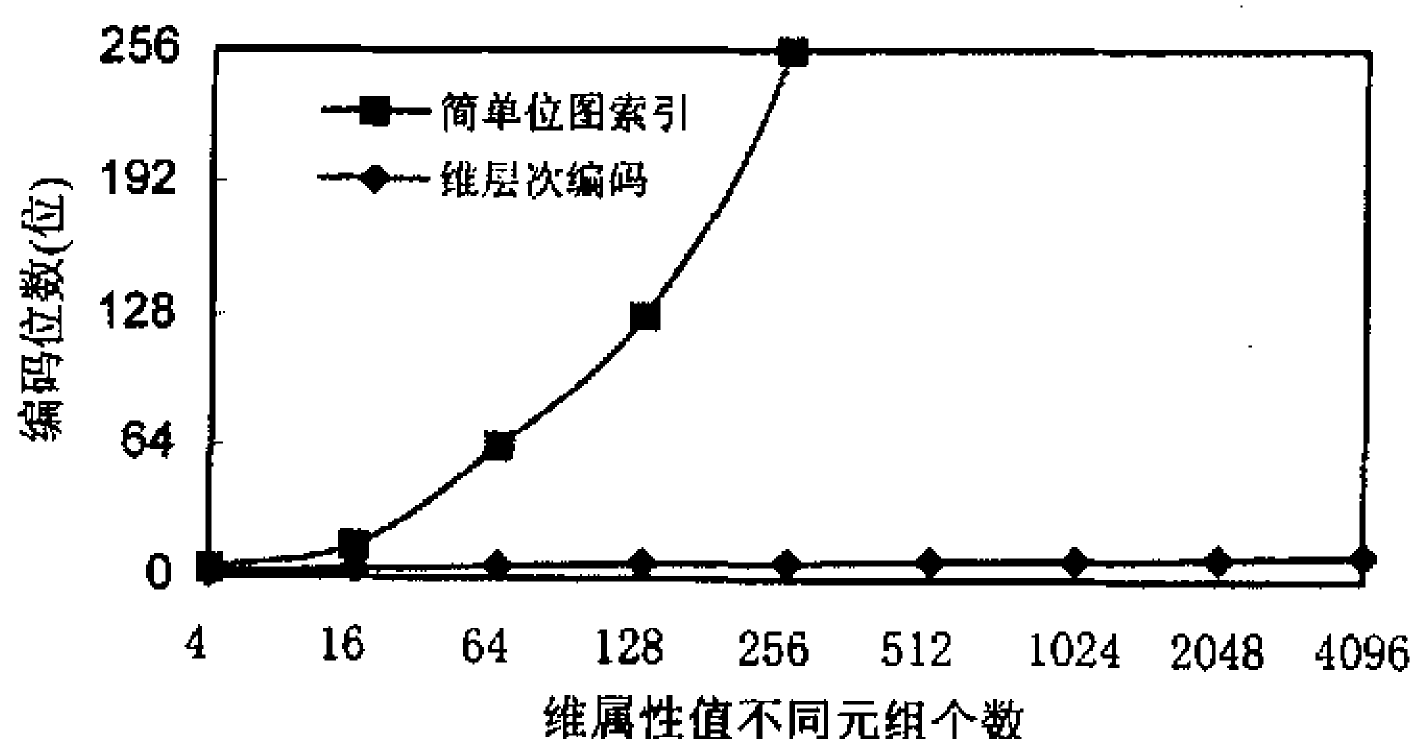


图3-4 维层次编码与简单位图索引的压缩性能

第2组实验测试是维层次编码中整数编码和二进制编码两种编码方式的压缩性能。测试数据为上述含有三个层次的产品维DimProduct，在其各个层次粒度中依次包含Ⅰ:10×10×100、Ⅱ:10×99×100、Ⅲ:10×100×100、Ⅳ:99×100×100、Ⅴ:100×100×100、Ⅵ：100×100×999六种不同个数的属性值。维层次编码中这二种编码方式需要的编码位数对比关系如图3-5所示。

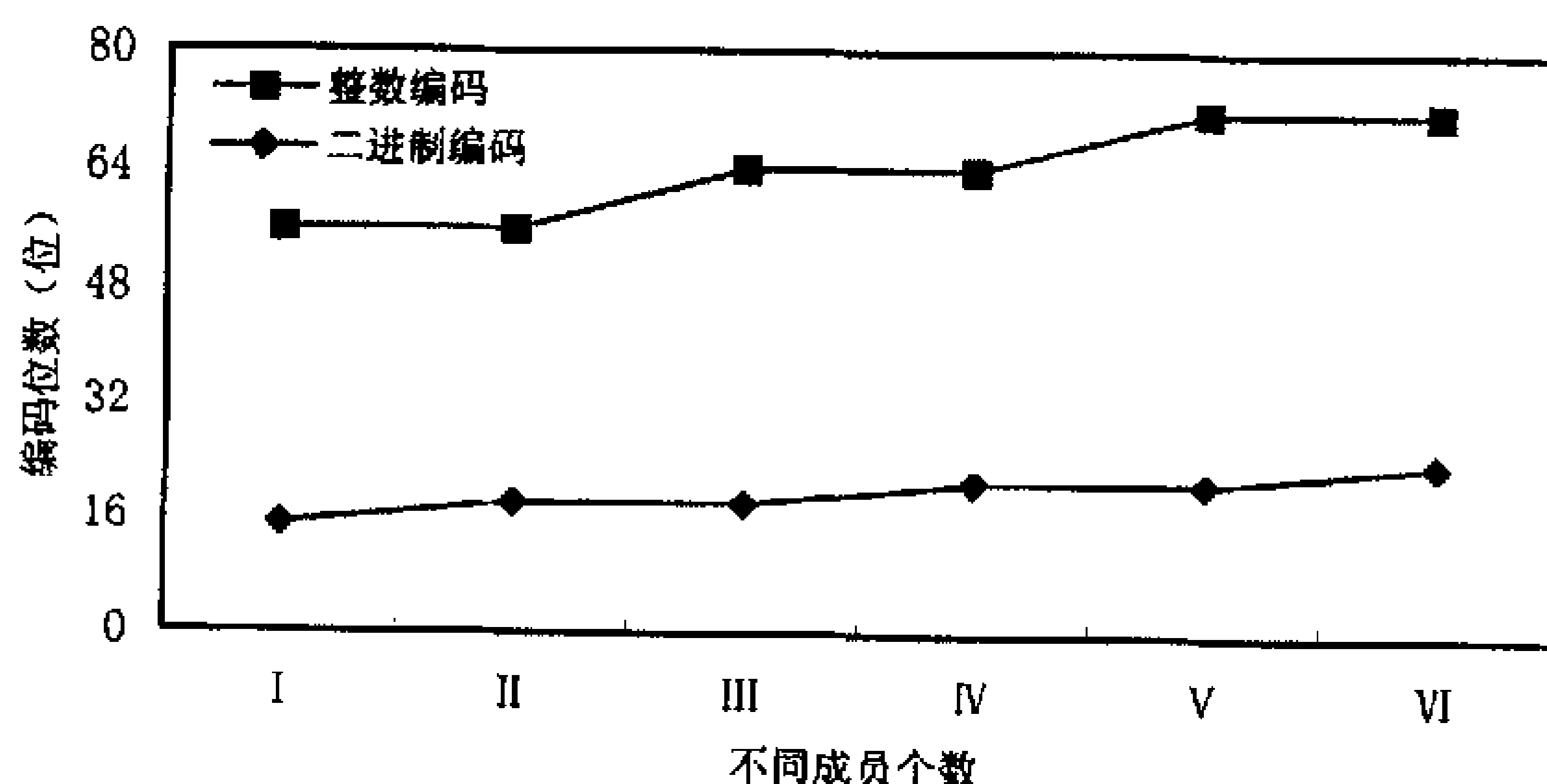


图3-5 维层次编码中两种编码方式的的压缩性能

由图3-4可以得出，若维属性成员个数为256时，则简单位图索引中的编码需要256位bits，而维层次编码只需要8位bits，数据压缩比为256/8=32。随着维属性成员个数继续增加时，其数据压缩比将成指数增长。由图3-5可以得出，在维层次编码的两种编码方式中，二进制编码方式要比整数编码方式数据平均压缩3~4倍。

为了进一步验证DHEPGA算法的优越性，我们采用维层次编码的DHEPGA算法和未采用维层次编码的基于分组序号和整数混合替代关键字的MuGA等算法进行查询操作比较分析。我们采用实验环境是基于Windows 2000 Server平台，实现环节采用Visual Studio作为开发工具，所采用的数据集为某企业ERP系统中的数据。其中具有代表性的几类典型数据查询性能如图3-6所示。

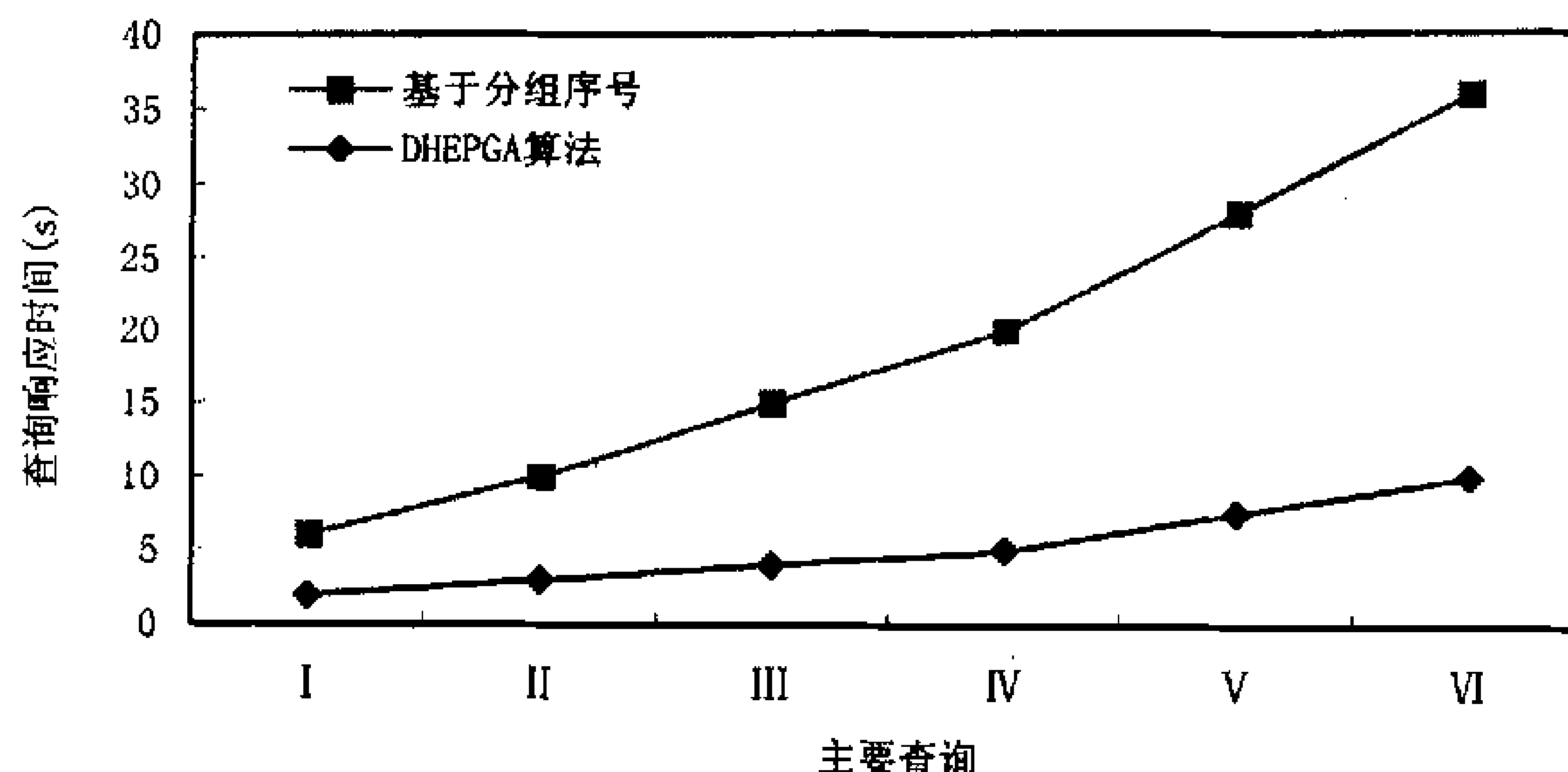


图 3-6 DHEPGA 算法的查询性能

图 3-6 表明，DHEPGA 算法是非常有效的。这是由于 DHEPGA 算法事先通过各个维的维层次编码对事实表中的数据进行簇集存储，将星形模式中的多表连接操作转换成为对维层次编码进行的范围查询，充分利用了维层次编码的层次性及编码前缀，对 OLAP 查询中的记录进行快速的分组聚集计算，所以 DHEPGA 算法性能有显著地提高。

3.5 本章小结

DHEPGA 算法主要利用了编码长度较小的维层次编码及其层次前缀路径，通过维层次前缀匹配操作，实现了在不访问维表记录情况下，检索出与关键字相匹配的维层次编码，来求得所有维层次属性的查询范围，提高检索速度，减少了 I/O 开销。充分利用维层次前缀对事实表中的数据进行快速预分组和层次聚集操作，实现了将 OLAP 查询中大量的多表连接转换为对维层次编码进行的范围查询，大大减少和简化了事实表与维表之间的多表连接，提高了 OLAP 查询效率。同时通过编码前缀提高了 roll up 和 drill down 等操作效率。算法分析和实验结果表明，相对于以往的分组聚集查询处理算法，DHEPGA 算法的性能有显著的提高。

第四章 维层次聚集 Cube

MOLAP 常常需在经过预聚集计算的聚集 Cube 上进行范围查询，从而使研究高性能的聚集 Cube 及其增量更新等相应算法成为提高 MOLAP 性能的关键。

本章首先介绍一些聚集 Cube 及其相应算法；然后针对这些聚集 Cube 在增量更新和压缩存储等方面的不足，提出利用 Cube 中的维层次聚集技术来创建高效的维层次聚集 Cube（DHAC）及其主要算法；最后对聚集 Cube 的性能进行分析和论证。

4.1 概述

在 MOLAP 中，数据通常以数据立方体（Data Cube）形式进行存储，通过数据聚集技术和高性能的多维存储结构组织和汇总了大量的聚集数据。MOLAP 常常需要在 Data Cube 上进行聚集范围查询，即在 Cube 上选取一个超立方块区域，计算出这个超立方块区域内单元数据的聚集值。对于人机交互的联机分析中，不论选取的查询范围是如何大，都必须对这些查询做出及时地响应。通常对 Data Cube 中这些区域的聚集值进行预先计算，并将结果保存到多维存储结构中，从而降低了从头开始重新计算聚集值的费用。但在 Data Cube 某个单元数据更新时，则对已进行预聚集运算的聚集 Cube 中受到更新影响的所有单元都要做出相应的更新，增加了更新费用。同时还需要额外的存储空间，增加了存储费用。从而使设计高效的聚集 Cube 及其增量更新等相应算法成为建立高性能多维数据仓库系统的关键。

本章结合近几年在数据仓库和 OLAP 等方面的研究成果，对高性能多维数据仓库系统进行了深入地研究，提出利用维层次树（B^+ Tree）为各个维建立索引，利用维层次聚集技术来创建维层次聚集 Cube（DHAC）。在用户提交 MOLAP 查询 Q 时，根据查询 Q 的查询空间 MBB_q 与 DHAC 中各结点的 MBB_c 进行比较，来计算 Q 的查询结果，从而提高了 MOLAP 查询效率。利用 Cube 的维层次聚集树中层次语义特性，沿着维层次聚集树由下而上用更新前后的差值，对受到更新影响的所有祖先结点进行增量更新，直到某一祖先结点不再更新为止，在最差情况下更新到根结点。即使在插入新维数据时，也不需要重新构建聚集 Cube，就可以对维层次聚集 Cube 进行增量更新，提高了聚集 Cube 的更新效率，实现了不需要重构 Cube 就可以进行维层次增删、维增删等模式更新操作。

4.2 聚集 Cube

在多维数据库中，数据立方体（Data Cube）通常用多维数组来存储数据。目前已有 PS、RPS、DDC 等多种多维数据存储方法，下面以聚集函数 SUM 为例，举例说明。

假设以存储数据仓库实视图或事实表的度量数据的二维数组 A 为例，其源数据立方体中部分数据如图 4-1 所示。

A	0	1	2	3	4	5	6	7	8
0	3	5	1	2	2	4	6	3	3
1	7	3	2	6	8	7	1	2	4
2	2	4	2	3	3	3	4	5	7
3	3	2	1	5	3	5	2	8	2
4	4	2	1	3	3	4	7	1	3
5	2	3	3	6	1	8	5	1	1
6	4	5	2	7	1	9	3	3	4
7	2	4	2	2	3	1	9	1	3
8	5	4	3	1	3	2	1	9	6

图 4-1　二维数据立方体数组 A

4.2.1 前缀和(Prefix Sum)

前缀和方法是对数据立方体 A 中各个单元的数据预先进行累加计算，并将累加的结果存储到与数组 A 同样大小的前缀和数组 PS 中。其计算公式见式 4.1，对二维数据立方体 A 进行前缀和聚集计算生成的 PS 如图 4-2 所示。

$$PS[x_1,x_2,\cdots,x_d]=Sum(A[0,0,\cdots,0]:A[x_1,x_2,\cdots,x_d])=\sum_{i_1=0}^{x_1}\sum_{i_2=0}^{x_2}\cdots\sum_{i_d=0}^{x_d}A[i_1,i_2,\cdots,i_d] \quad (4.1)$$

式中：$PS[x_1,x_2,\cdots,x_d]$为前缀和数组；

$A[0,0,\cdots,0]$，$A[x_1,x_2,\cdots,x_d]$为源数据立方体数组；

$x_1,x_2,\cdots,x_d$ 分别为数组的下标值（以下类同）。

PS	0	1	2	3	4	5	6	7	8
0	3	8	9	11	13	17	23	26	29
1	10	18	21	29	39	50	57	62	69
2	12	24	29	40	53	67	78	88	102
3	15	29	35	51	67	86	99	117	133
4	19	35	42	61	80	103	123	142	161
5	21	40	50	75	95	126	151	171	191
6	25	49	61	93	114	154	182	205	229
7	27	55	69	103	127	168	205	229	256
8	32	64	81	116	143	186	224	257	290

图 4-2　前缀和 PS

在用户提出查询时，根据 PS 数组即可很快的查得结果，查询效率较高。但是在数据更新时，其维护费用非常大，如果数据立方体 A 中某一处的数据进行了修改，则数组 PS 中在该处后面的所有数据单元都需要进行修改和重新计算，产生了数据的串联更新。

4.2.2 相对前缀和(Relative Prefix Sum)

为了减少前缀和方法的数据更新维护费用，采用相对前缀和方法，即将数据立方体分成若干小方块，在每一个方块中分别进行前缀和计算，从而避免前缀和方法中的串联更新。

本方法使用到相对前缀和 *RPS*、相对重叠（Relative Overlay）数组 *RO* 和块前缀(Block Prefix)数组 *BP* 三个数据结构，其计算公式分别见式 4.2、式 4.3 和式 4.4。

$$RPS[x_1,x_2,\cdots,x_d]=Sum(A[a_1,a_2,\cdots,a_d]:A[x_1,x_2,\cdots,x_d])=\sum_{i_1=a_1}^{x_1}\sum_{i_2=a_2}^{x_2}\cdots\sum_{i_d=a_d}^{x_d}A[i_1,i_2,\cdots,i_d] \quad (4.2)$$

$$V_{RO}=Sum(A[a_1,a_2,\cdots,a_d]:A[a_1+k_1,a_2+k_2,\cdots,a_d+k_d])=\sum_{i_1=a_1}^{a_1+k_1}\sum_{i_2=a_2}^{a_2+k_2}\cdots\sum_{i_d=a_d}^{a_d+k_d}A[i_1,i_2,\cdots,i_d] \quad (4.3)$$

$$V_{BP}=Sum(A[0,0,\cdots,0]:A[a_1-1,a_2-1,\cdots,a_d-1])=\sum_{i_1=0}^{a_1-1}\sum_{i_2=0}^{a_2-1}\cdots\sum_{i_d=0}^{a_d-1}A[i_1,i_2,\cdots,i_d] \quad (4.4)$$

式中：$RPS[x_1,x_2,\cdots,x_d]$ 为相对前缀和数组；

$(a_1,a_2,\cdots,a_d)$ 是该块框（Block Box）中最小索引单元；

$(k_1,k_2,\cdots,k_d)$ 是该块框（Block Box）中最大索引单元的偏址；

V_{RO} 是每个重叠框单元的累计聚集值；

V_{BP} 是块框单元的累计聚集值。

以上述的二维数组 A 为例，数组 *RPS* 的计算结果如图 4-3 所示，数组 *RO* 是图 4-4 中的未作底纹的部分、数组 *BP* 是图 4-4 中作了底纹的部分。

RPS	0	1	2	3	4	5	6	7	8
0	3	8	9	2	4	8	6	9	12
1	10	18	21	8	18	29	7	12	19
2	12	24	29	11	24	38	11	21	35
3	3	5	6	5	8	13	2	10	12
4	7	11	13	8	14	23	9	18	23
5	9	16	21	14	21	38	14	24	30
6	4	9	11	7	8	17	3	6	10
7	6	15	19	9	13	23	12	16	23
8	11	24	31	10	17	29	13	26	39

图 4-3 相对前缀和 RPS

	0	0	0	0	0	0	0	0	0	0	0	0
0				9				17				29
0				21				50				69
0				29				67				102
	12	24	29	29	11	24	38	67	11	21	35	102
0				5				19				31
0				13				36		*		59
0				21				59				89
	21	40	50	50	25	45	76	126	25	45	65	191
0				11				28				38
0				19				42				65
0				31				60				99
	32	64	81	81	35	62	105	186	37	71	104	290

图 4-4 数组 RO 和数组 BP

对 PS 中各个单元的聚集值可以由 V_{RO}、V_{BP} 与 RPS 中相应单元的聚集值进行聚集计算得到，即：

$$PS[x_1, x_2, \cdots, x_d] = V_{RO} + V_{BP} + RPS[x_1, x_2, \cdots, x_d] \tag{4.5}$$

例如对图 4-4 中的*单元的计算为 67+21+36+18=142，正好等于数组 $PS[4][7]$单元的聚集值。

4.2.3 有效空间相对前缀和 SRPS (Space-Efficient Relative Prefix Sum)

有效空间相对前缀和方法与前面的相对前缀和方法相类似，只不过它没有额外地利用相对前缀和方法中的数组 RO 和 BP，而是将这些数组中的信息都保存在同一个数组 $SRPS$ 中。其计算公式见式 4.6 所示。与上述数组 A 相对应的 $SRPS$ 如图 4-5 所示。

$$SRPS[x_1, x_2, \cdots, x_d] = Sum(A[l_1, l_2, \cdots, l_d] : A[x_1, x_2, \cdots, x_d]) \tag{4.6}$$

式中：如果 $x_i = a_i$ 则 l_i=0，否则如果 $a_i+1 \leq x_i < a_i+k$ 则 $l_i=a_i+1$；

k 为每个子块的单元个数，a_i 同前。

SRPS	0	1	2	3	4	5	6	7	8
0	3	5	6	11	2	6	23	3	6
1	7	3	5	18	8	15	34	2	6
2	9	7	11	29	11	21	55	7	18
3	15	14	20	51	16	35	99	18	34
4	4	2	3	10	3	7	24	1	3
5	6	5	9	24	4	16	52	2	6
6	25	24	36	93	21	61	182	23	47
7	2	4	6	10	3	4	23	1	3
8	7	8	13	23	6	9	42	10	19

图 4-5 有效空间相对前缀和 SRPS

4.2.4 动态数据立方体 DDC (Dynamic Data Cube)

DDC 是利用树递归的方法将数组 A 划分成若干重叠框（Overlay Box），每个重叠框包含数组 A

各个分块的相对累加值。在 DDC 中，由树根向下进行累加相应单元的聚集函数值，即可得出 A 中从 A[0][0]单元到任一单元的聚集函数值，来计算 Cube 中的聚集值。如 PS 中的有底纹的单元 *PS*[4][5]是由 DDC 中有底纹的各个单元进行聚集计算得到，即 *PS*[4][5]=51+35+10+7=103。

A	0	1	2	3	4	5	6	7
0	3	5	1	2	2	4	6	3
1	7	3	2	6	8	7	1	2
2	2	4	2	3	3	3	4	5
3	3	2	1	5	3	5	2	8
4	4	2	1	3	3	4	7	1
5	2	3	3	6	1	8	5	1
6	4	5	2	7	1	9	3	3
7	2	4	2	2	3	1	9	1

数组 A

PS	0	1	2	3	4	5	6	7
0	3	8	9	11	13	17	23	26
1	10	18	21	29	39	50	57	62
2	12	24	29	40	53	67	78	88
3	15	29	35	51	67	86	99	117
4	19	35	42	61	80	103	123	142
5	21	40	50	75	95	126	151	171
6	25	49	61	93	114	154	182	205
7	27	55	69	103	127	168	205	229

			11				15
			29				33
			40				48
15	29	35	51	16	35	48	66
			10				15
			24				30
			42				46
12	26	34	52	8	30	54	60

	8		3
10	18	3	11
	6		5
5	11	3	11

	6		9
10	21	7	12
	6		9
6	14	6	19

	6		4
6	11	4	13
	9		9
6	15	4	13

	[illegible]		8
4	16	12	14
	10		6
4	14	12	16

3 5 / 7 3　　1 2 / 2 6　　2 4 / 8 7　　6 3 / 1 2

2 4 / 3 2　　2 3 / 1 5　　3 3 / 3 5　　4 5 / 2 8

4 2 / 2 3　　1 3 / 3 6　　3 4 / 1 8　　7 1 / 5 1

4 5 / 2 4　　2 7 / 2 2　　1 9 / 3 1　　3 3 / 9 1

DDC 树：Level 2(Root),K=4　　Level 1(Root),K=2　　Level 0 (leaves),K=1

图 4-6 动态数据立方体 DDC

4.2.5 有效空间动态数据立方体 SDDC (Space-Efficient Dynamic Data Cube)

SDDC 同 DDC 一样进行划分，只不过在树中下一层中不再保存相对前缀累加信息，而是将各个层次聚集数据存储到同一个聚集 Cube 中。其查询同 DDC 相类似。

A	0	1	2	3	4	5	6	7	8	9
0	3	5	1	2	2	4	6	3	3	1
1	7	3	2	6	8	7	1	2	4	2
2	2	4	2	3	3	3	4	5	7	4
3	3	2	1	5	3	5	2	8	2	1
4	4	2	1	3	3	4	7	1	3	2
5	2	3	3	6	1	8	5	1	1	2
6	4	5	2	7	1	9	3	3	4	1
7	2	4	2	2	3	1	9	1	3	3
8	5	4	3	1	3	2	1	9	6	5
9	6	1	2	4	2	1	3	1	5	2

数组 A

SDDC	0	1	2	3	4	5	6	7	8	9
0	3	5	6	8	10	17	6	9	12	13
1	7	3	2	11	8	33	1	2	7	2
2	9	4	2	3	3	50	4	5	7	4
3	12	9	5	28	14	69	7	15	35	7
4	16	2	1	6	3	86	7	1	11	2
5	18	19	29	54	74	126	25	45	65	77
6	4	5	2	14	1	28	3	3	10	1
7	6	4	2	8	3	42	9	1	13	3
8	11	13	7	30	7	60	13	9	39	9
9	17	1	2	7	2	76	3	1	9	2

SDDC

图 4-7 有效空间动态数据立方体 SDDC

4.2.6 聚集 Cube 主要操作

可以采用迭代计算法来实现在上述聚集 Cube 上进行的创建、查询、更新等操作。

1）聚集 Cube 的创建

可以通过迭代的方法依次对各个维采用相应预集聚技术进行聚集运算，来创建聚集 Cube。聚集 Cube 可以按式 4.7 进行迭代聚集运算来创建，其创建算法如算法 4.1 所示。

$$A_1[c_1,c_2,\cdots,c_d]=\sum_{k_1=0}^{n_1-1}\alpha_{1,c_1,k_1}A[k_1,c_2,\cdots,c_d]$$

$$A_2[c_1,c_2,\cdots,c_d]=\sum_{k_2=0}^{n_2-1}\alpha_{2,c_2,k_2}A_1[k_1,k_2,c_3,\cdots,c_d]=\sum_{k_1=0}^{n_1-1}\sum_{k_2=0}^{n_2-1}\alpha_{1,c_1,k_1}\alpha_{2,c_2,k_2}A[k_1,k_2,c_3,\cdots,c_d]$$

……

$$A_d[c_1,c_2,\cdots,c_d]=\sum_{k_1=0}^{n_1-1}\sum_{k_2=0}^{n_2-1}\cdots\sum_{k_d=0}^{n_d-1}\alpha_{1,c_1,k_1}\alpha_{2,c_2,k_2}\cdots\alpha_{d,c_d,k_d}A[k_1,k_2,\cdots,k_d] \tag{4.7}$$

式中：α_{i,c_i,k_i} 是由第 i 维所采用的预聚集技术 Θ_i 所决定。

算法 4.1：聚集 Cube 的创建算法

Input: original data cube A, one-dimensional techniques Θ_1, Θ_2, …, Θ_d

Output: pre-aggregated data cube A_d

```
{A_0=A;
  For (each dimension δ_i )
     {For each(c_1∈{0, 1, …, n_1-1}, …, c_{i-1}∈{0, 1, …, n_{i-1}-1}, c_{i+1}∈{0, 1, …, n_{i+1}-1}, …, c_d∈{0, 1, …, n_d-1})
          (A_i[c_1,…,c_{i-1},0,c_{i+1},…,c_d],A_i[c_1,…,c_{i-1},1,c_{i+1},…,c_d],…, A_i[c_1,…,c_{i-1},n_i-1,c_{i+1},…,c_d])
          = Θ_i.Construct(A_{i-1}[c_1,…,c_{i-1},0,c_{i+1},…,c_d],A_{i-1}[c_1,…,c_{i-1},1,c_{i+1},…,c_d],…, A_{i-1}[c_1,…,c_{i-1},n_i-1,c_{i+1},…,c_d]);
     }
}
```

Function Θ_i.Construct

Input: array B with n values $B[0],B[1],\cdots,B[n-1]$

Output: corresponding pre-aggregated array B_i

{ For(j=0 to n-1)

$$B_i[j]=\sum_{k=0}^{n-1}\alpha_{i,j,k}B[k];$$

Return($B_i[0],B_i[1],\cdots,B_i[n-1]$);}

}

2）聚集 Cube 的查询

聚集 Cube 可以按式 4.8 进行聚集查询计算。其查询算法如算法 4.2 所示。

$$Q=\sum_{j_d\in r_d}\sum_{j_{d-1}\in r_{d-1}}\cdots\sum_{j_2\in r_2}\left(\sum_{l_1=0}^{n_1-1}\beta_{1,r_1,l_1}A_1[l_1,j_2,\cdots,j_d]\right)=\sum_{l_1=0}^{n_1-1}\sum_{l_2=0}^{n_2-1}\cdots\sum_{l_d=0}^{n_d-1}\beta_{1,r_1,l_1}\beta_{2,r_2,l_2}\cdots\beta_{d,r_d,l_d}A_d[l_1,l_2,\cdots,l_d] \tag{4.8}$$

式中：β_{i,r_i,l_i} 是由第 i 维所采用的预聚集技术 Θ_i 和查询的范围 r_i 来决定。

算法 4.2：聚集 Cube 的查询算法

Input: pre-aggregrated data cube A_d; ranges $r_1,r_2,\cdots,r_d$

Output: range *sum* for selected range on A

{ *Result*=0;

For (each dimension δ_i)

$S_i=\Theta_i.QueryRange(r_i)$;

For (each (β_{1,r_1,l_1}, l_1)$\in S_1$)

For (each (β_{2,r_2,l_2}, l_2)$\in S_2$)

…

For (each (β_{d,r_d,l_d}, l_d)$\in S_d$)

$Result=Result+\beta_{1,r_1,l_1}\beta_{2,r_2,l_2}\cdots\beta_{d,r_d,l_d}A_d[l_1,l_2,\cdots,l_d]$;

Return(*Result*);

}

Function Θ_i**.*QueryRange***

Input: range r

Output: translated “range” with scaling factors,i.e.,$\{(\beta_{i,r,l},l) \mid \beta_{i,r,l}\neq 0\}$

{ $S=\{\}$;

For(all l)

If ($\beta_{i,r,l}\neq 0$) then $S=S\cup\{(\beta_{i,r,l},l)\}$;

Retrun(S);}

3）聚集 Cube 的更新

如果源数据立方体 A 中的某一个单元更新值为△，则对聚集 Cube 中所有 $\alpha_{1,c_1,k_1}\alpha_{2,c_2,k_2}\cdots\alpha_{d,c_d,k_d}$ $\neq 0$ 的单元 $A_d[c_1,c_2,\cdots,c_d]$ 用更新值△ $\cdot\,\alpha_{1,c_1,k_1}\alpha_{2,c_2,k_2}\cdots\alpha_{d,c_d,k_d}$ 来进行更新。其更新算法如算法 4.3 所示。

算法 4.3：聚集 Cube 的更新算法

Input: pre-aggregrated data cube A_d; updated cell $u=[u_1,\cdots,u_d]$ in original data cube;

Difference △ between new and old value of u

Output: Updated data cube A_d

{ For (each dimension δ_i)

$S_i=\Theta_i.UpdateRange(u_i)$;

For (each (α_{1,c_1,u_1}, c_1)$\in S_1$)

For (each (α_{2,c_2,u_2} , c_2) $\in S_2$)

…

For (each (α_{d,c_d,u_d} , c_d) $\in S_d$)

$$A_d[c_1,c_2,\cdots,c_d]=A_d[c_1,c_2,\cdots,c_d]+\alpha_{1,c_1,u_1}\alpha_{2,c_2,u_2}\cdots\alpha_{d,c_d,u_d}\bullet\Delta;$$

Return(A_d);}

Function Θ_i.***UpdateRange***

Input: index u_i of updated cell

Output: the range of affected cells' indeces with scaling factors,i.e., $\{(\alpha_{i,j,u_i},j)|\alpha_{i,j,u_i}\neq 0\}$

{ $S=\{\}$;

For all j do

If ($\alpha_{i,j,u_i}\neq 0$) then $S=S\cup\{(\alpha_{i,j,u_i},j)\}$;

Retrun(S); }

4.3 维层次聚集 Cube

定义 4. 1(立方体 Cube). 在数据仓库中，Cube 可以用 $C=(D, M)$ 来表示。其中 $D=\{D_1, \cdots, D_m\}$ 是维的集合，$m\geq 1$；$M=\{M_1, \cdots, M_n\}$ 是度量属性的集合，$n\geq 1$。

其中：$D_i=(DH_i, \preceq)$ 表示一个维，DH_i 表示维 D_i 的所有层次所组成的一个有序集，即 $DH_i=(L_1^i,L_2^i,\cdots,L_h^i)$，$L_i^i$ 是维的层次，$\preceq$ 是 DH_i 中维层次的依赖关系或偏序关系。$\forall$ L_i^i、$L_j^i\in DH_i$ 且 $L_i^i\preceq L_j^i$，则 L_i^i 层的层次比层 L_j^i 高，即 L_i^i 层的层次粒度比层 L_j^i 的要粗。

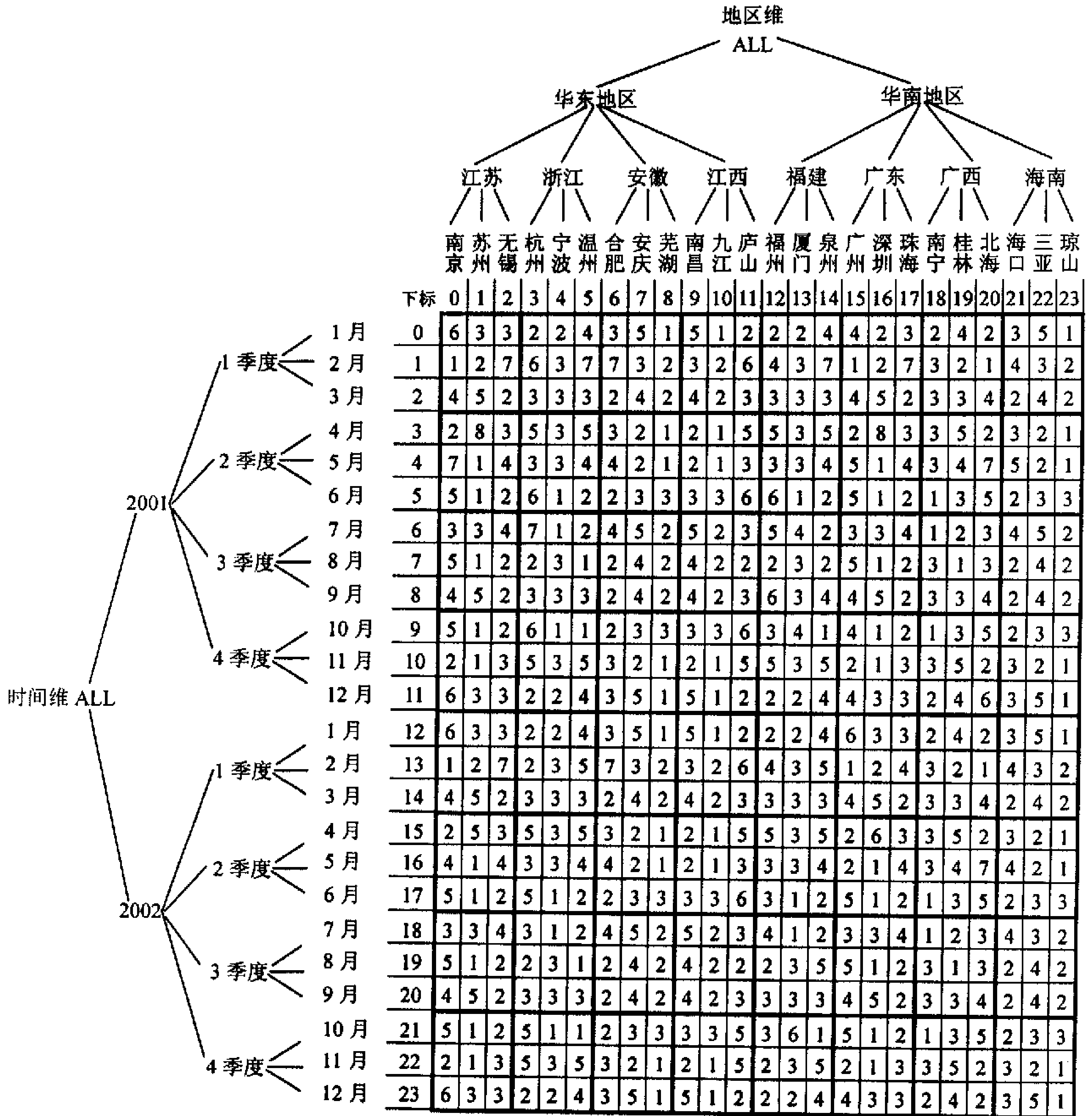
定义 4. 2（维层次 B$^+$树） 利用维层次 B$^+$树（Dimension Hierarchy B$^+$ Tree，简称 DHB$^+$ Tree）为 Cube 中的每一个维建立索引，整个 Cube 需要 d 棵维层次 B$^+$树（DHB$^+$ Tree）来进行索引，这 d 棵维层次 B$^+$树相互独立，每一棵 DHB$^+$ Tree 存储一个对应维的成员。维层次 B$^+$树定义如下：

（1）DHB$^+$ Tree 中的各层结点存储了对应维的维成员信息。

（2）最低层次的叶子结点存储了维最细层次粒度的明细信息，通过指针标明了这个结点在维中的位置。

（3）同一个层次中的叶子结点都用指针连接起来，每一层次的 B+树阶数可以不同，视具体情况而定。

某销售数据仓库中的二维 Cube 及其时间维、地区维的维层次 B+树如图 4-8 所示。



		南京	苏州	无锡	杭州	宁波	温州	合肥	安庆	芜湖	南昌	九江	庐山	福州	厦门	泉州	广州	深圳	珠海	南宁	桂林	北海	海口	三亚	琼山
	下标	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1 月	0	6	3	3	2	2	4	3	5	1	5	1	2	2	2	4	4	2	3	2	4	2	3	5	1
2 月	1	1	2	7	6	3	7	7	3	2	3	2	6	4	3	7	1	2	7	3	2	1	4	3	2
3 月	2	4	5	2	3	3	3	2	4	2	4	2	3	3	3	3	4	5	2	3	3	4	2	4	2
4 月	3	2	8	3	5	3	5	3	2	1	2	1	5	5	3	5	2	8	3	3	5	2	3	2	1
5 月	4	7	1	4	3	3	4	4	2	1	2	1	3	3	3	4	5	1	4	3	4	7	5	2	1
6 月	5	5	1	2	6	1	2	2	3	3	3	3	6	6	1	2	5	1	2	1	3	5	2	3	3
7 月	6	3	3	4	7	1	2	4	5	2	5	2	3	5	4	2	3	3	4	1	2	3	4	5	2
8 月	7	5	1	2	2	3	1	2	4	2	4	2	2	2	3	2	5	1	2	3	1	3	2	4	2
9 月	8	4	5	2	3	3	3	2	4	2	4	2	3	6	3	4	4	5	2	3	3	4	2	4	2
10 月	9	5	1	2	6	1	1	2	3	3	3	3	6	3	4	1	4	1	2	1	3	5	2	3	3
11 月	10	2	1	3	5	3	5	3	2	1	2	1	5	5	3	5	2	1	3	3	5	2	3	2	1
12 月	11	6	3	3	2	2	4	3	5	1	5	1	2	2	2	4	4	3	3	2	4	6	3	5	1
1 月	12	6	3	3	2	2	4	3	5	1	5	1	2	2	2	4	6	3	3	2	4	2	3	5	1
2 月	13	1	2	7	2	3	5	7	3	2	3	2	6	4	3	5	1	2	4	3	2	1	4	3	2
3 月	14	4	5	2	3	3	3	2	4	2	4	2	3	3	3	3	4	5	2	3	3	4	2	4	2
4 月	15	2	5	3	5	3	5	3	2	1	2	1	5	5	3	5	2	6	3	3	5	2	3	2	1
5 月	16	4	1	4	3	3	4	4	2	1	2	1	3	3	3	4	2	1	4	3	4	7	4	2	1
6 月	17	5	1	2	5	1	2	2	3	3	3	3	6	3	1	2	5	1	2	1	3	5	2	3	3
7 月	18	3	3	4	3	1	2	4	5	2	5	2	3	4	1	2	3	3	4	1	2	3	4	3	2
8 月	19	5	1	2	2	3	1	2	4	2	4	2	2	2	3	5	5	1	2	3	1	3	2	4	2
9 月	20	4	5	2	3	3	3	2	4	2	4	2	3	3	3	3	4	5	2	3	3	4	2	4	2
10 月	21	5	1	2	5	1	1	2	3	3	3	3	5	3	6	1	5	1	2	1	3	5	2	3	3
11 月	22	2	1	3	5	3	5	3	2	1	2	1	5	2	3	5	2	1	3	3	5	2	3	2	1
12 月	23	6	3	3	2	2	4	3	5	1	5	1	2	2	2	4	4	3	3	2	4	2	3	5	1

图 4-8 二维 Cube 及其维层次树

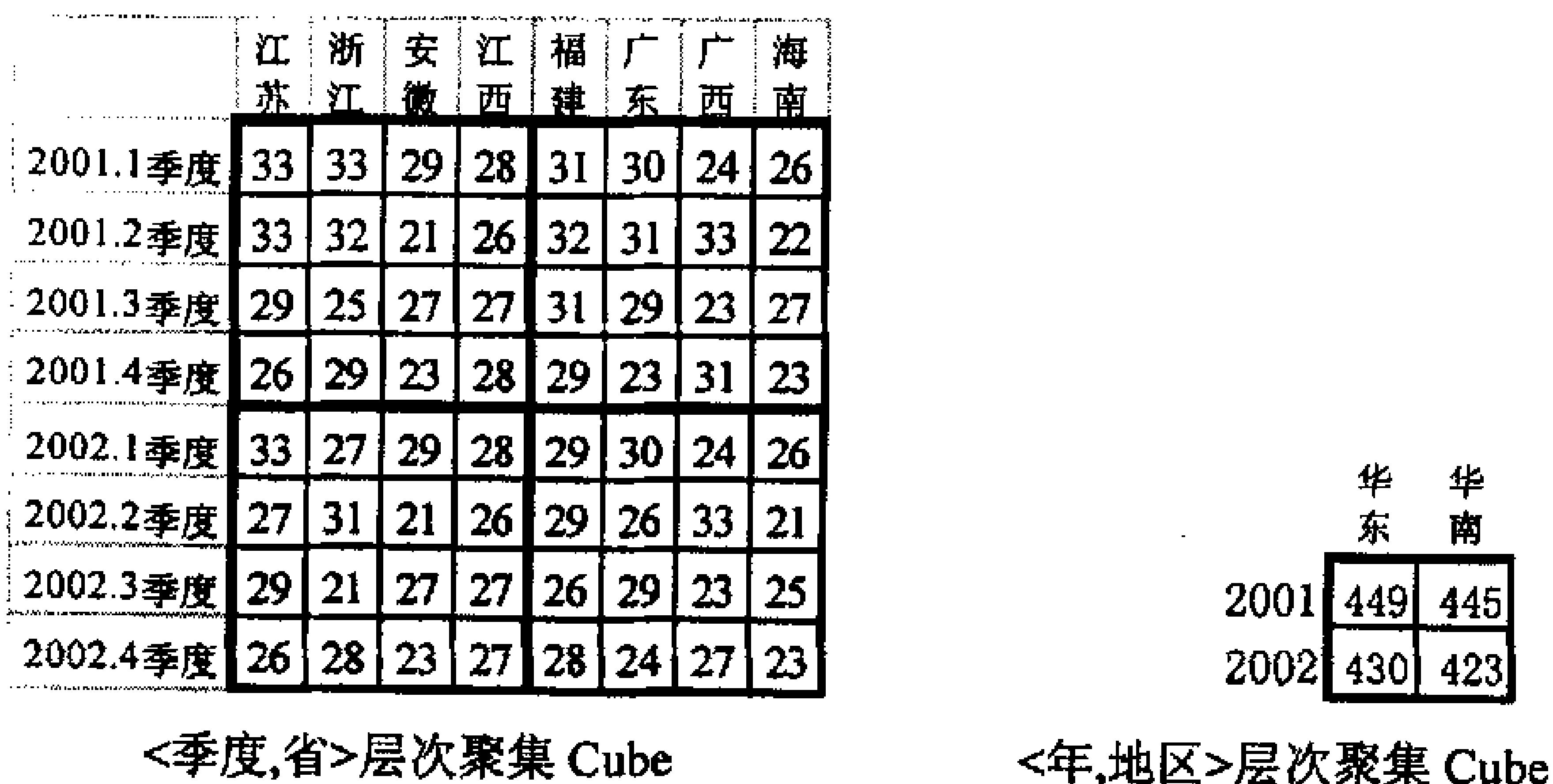
定义 4.3(维层次组合). 对于一个 d-维的 Cube，第 i 维有 H_i 个层次($1 \leqslant i \leqslant d$)，称 $<h_1,h_2,\cdots\cdots,h_d>$ 为 Cube 上的一个维层次组合，其中 $1 \leqslant h_k \leqslant H_k$ ($1 \leqslant i \leqslant d$)。所有维层次组合构成维层次组合集 DH。在 DH 上的 $\preceq$ 运算为：$<a_1,a_2,\cdots\cdots,a_d> \preceq <b_1,b_2,\cdots\cdots,b_d>$，当且仅当对所有的 i($1 \leqslant i \leqslant d$)，都有 $a_i \preceq b_i$。

例 4.1 对于一个三维的多维数据集<时间，地区，产品>，其中第一个维为时间维，有年、季度、月三个层次；第二个维为地区维，有地区、省、市三个层次；第三维为产品维，有产品大类、产品小类、产品三个层次；则<年,地区,产品大类>、<季度,省,产品小类>，<月,市,产品>等都是 Cube 上的维层次组合。

对于维层次组合 $<h_1,h_2,\cdots\cdots,h_d>$，如果在第 i 维上依照 h_i 进行逻辑划分，并且对于每个逻辑块都根据维层次 B+树的第 h_i 层进行块内聚集并存储到上一层次的聚集 Cube 中，则称得到的 Cube 为维

层次组合<$h_1,h_2,\cdots\cdots,h_d$>所对应的层次聚集 Cube。

例 4.2 对图 4-8 所示的维层次组合为<月,市>的二维 Cube，分别按其时间维、地区维的季度和省层次进行逻辑划分，对每一个划分的逻辑块按维层次树的层次进行块内聚集计算，将聚集值存储到上一个维层次组合<季度,省>的层次聚集 Cube 中存储起来。依次对维层次组合<季度,省>的层次聚集 Cube 按年和地区层次进行逻辑划分和聚集计算，生成维层次组合<年,地区>的层次聚集 Cube。其层次聚集 Cube 逻辑划分如图 4-9 所示。



	江苏	浙江	安徽	江西	福建	广东	广西	海南
2001.1季度	33	33	29	28	31	30	24	26
2001.2季度	33	32	21	26	32	31	33	22
2001.3季度	29	25	27	27	31	29	23	27
2001.4季度	26	29	23	28	29	23	31	23
2002.1季度	33	27	29	28	29	30	24	26
2002.2季度	27	31	21	26	29	26	33	21
2002.3季度	29	21	27	27	26	29	23	25
2002.4季度	26	28	23	27	28	24	27	23

<季度,省>层次聚集 Cube

	华东	华南
2001	449	445
2002	430	423

<年,地区>层次聚集 Cube

图 4-9 维层次聚集 Cube 逻辑划分

在物理存储的时候，我们可以对整个 Cube 多维数据集依照各个维的层次进行逻辑划分，首先对各个维的最低层次进行逻辑划分，所得到的各个逻辑块可以根据实际情况分别存储。逻辑块存储在磁盘时可以作为一个整体，也可以再次划分为若干个具有固定大小的物理块进行存储。

为了提高聚集查询的效率，需要对数据进行预聚集，即将多维数据集划分并生成不同的层次聚集 Cube。在生成层次聚集 Cube 时，对于某个聚集 Cube 而言，如果它至少有一个直接后继层次聚集 Cube，那么它的聚集值计算就可以从这个直接后继的层次聚集 Cube 中沿各个维的维层次 B+树向上进行聚集计算得到，而不用去访问层次粒度最明细的元组，即不需要访问源 Cube 中的数据。

定义 4.4(搜索范围 MBB). 对于 d 维空间，Cube 的搜索范围可以用最小限定盒（Minimum Bounding Box）$MBB_c=\{c_1,c_2,\cdots c_d\}$ 来表示。

其中，c_i 用一个二元组(cx_i,cy_i)来表示，cx_i、cy_i分别为 MBB_c在 i 维的最小和最大编码值，即第 i 维的编码取值范围。cx_i、cy_i 可以用整数编码和二进制编码来表示，为了对编码进行压缩，这里考虑采用经过压缩的长度较小的维层次编码来表示，参见定义 3.6 维层次编码。

定义 4.5(维层次聚集 Cube,简称 DHAC). 维层次聚集 Cube 是通过 Cube 中各个维层次树中的维层次属性，对 Cube 按各个维层次进行分割和分组聚集计算生成的一种具有层次性的 Cube。维层次聚集 Cube 主要是用具有以下特点的维层次聚集树来进行存储：

(1) 维层次聚集 Cube 中含有根结点、目录结点和叶子结点三种结点，根结点是一个特殊的目录结点。

(2)维层次聚集 Cube 中各结点具有层次性，即每一层 Cube 中各个维的维层次要比下一层 Cube（即子 Cube）的维层次粒度粗，第一层 Cube（即根结点 Cube）的维层次粒度是最粗的。

(3) 维层次聚集 Cube 中的结点为 *DHAC_TreeNode*<*flag,childPtr, parentPtr,* MBB_c*,Chunk*>，当 *flag*=1 时为目录结点，*flag*=0 时为叶结点；其中 MBB_c 表示该结点所包含的范围，*childPtr* 为指向下一层子 Cube 的指针，*parentPtr* 为指向上一层父 Cube 的指针，*Chunk* 主要是用于存储具有相同维层次的多个数据单元 *cell*<*address,AGG*>组成，其大小以能够完全被

装入内存为宜。

（4）根结点和目录结点的 *MBB* 包含或覆盖范围大小等于它的所有孩子结点的 *MBB* 之和；根结点和目录结点中各数据单元 *cell* 的 *AGG* 是其所包含的孩子结点中相应 *Chunk* 中所有数据单元 *cell* 的 *AGG* 进行聚集计算的聚集值。

图 4-8 中二维 Cube 的 DHAC 层次存储结构如图 4-10 所示。

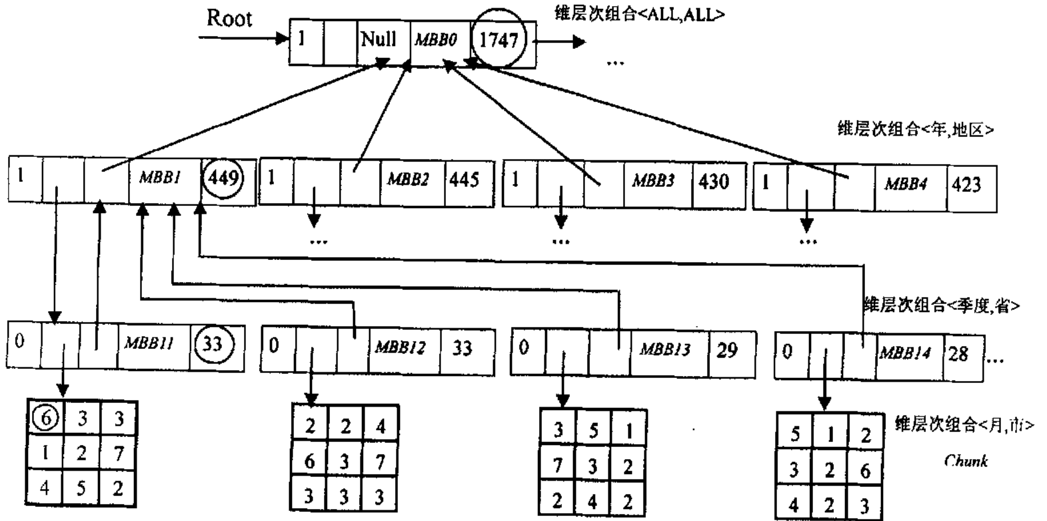


图 4-10 DHAC 的层次存储结构

定义 4.6（父立方体、孩子立方体）．设 $Cuboid_i$、$Cuboid_j$ 是在维层次聚集 Cube 中的两个分割子 Cube，如果 $Cuboid_i$ 是由 $Cuboid_j$ 聚集计算而来（即 $Cuboid_j.MBB \subset Cuboid_i.MBB$、$Cuboid_i.L_i^l \preceq Cuboid_j.L_j^l$，$Cuboid_i$ 中各个维的维层次粒度比 $Cuboid_j$ 中的要粗），则称 $Cuboid_i$ 与 $Cuboid_j$ 是父子关系，$Cuboid_i$ 为 $Cuboid_j$ 的父 Cube，$Cuboid_j$ 为 $Cuboid_i$ 的孩子 Cube。

4.4 DHAC 主要算法

4.4.1 DHAC 的创建算法

通过数据仓库建模工具为 Cube 中的各个维创建维层次树，从维层次粒度最细的 Cube 开始，按各个维层次树中的维层次属性，对 Cube 中粒度较细的各个数据单元（Cell）进行划分成若干个互不相交的分割子 Cube，并对每一个分割子 Cube 的聚集值按维层次前缀进行分组聚集计算，将计算得到的聚集值存储到父 Cube 中，再对生成的父 Cube 按维层次粒度由细到粗进行递归地分割和分组聚集计算，直至维层次粒度最粗时为止。分割时一般以较粗粒度的维层次成员作为划分的分割线，尽量将具有同一个较粗粒度层次维成员的各个细粒度的维成员划分到同一个子 Cube 中。

其创建算法形式描述见算法 4.4。

算法 4.4：DHAC *CreateDHAC* (DHAC $Cuboid_i$)

{ if($Cuboid_i$ 中各个维的维层次不是最粗粒度)

{将 $Cuboid_i$ 按维层次树第 i-1 层的维层次属性进行分割成若干个子 $Cuboid_{i,j}$（假设为 M 个,j=1,…,M);

建立含 M 个节点的 $Cuboid_{i-1}$，将 $childPtr_j$ 分别指向这 M 个分割的子 $Cuboid_{i,j}$;

将子立方体 $Cuboid_{i,j}$ 的 $parentPtr_j$ 指向 $Cuboid_{i-1}$ 中的第 j 节点；

for(j=0;j<M;j++)

{计算子立方体 $Cuboid_{i,j}$ 的聚集值 $Cuboid_{i,j}.AGG$ 和子立方体 $Cuboid_{i,j}$ 的聚集范围 $MBB_{i,j}$，一并保存到 $Cuboid_{i-1}$ 中相应的节点及其 *Chunk* 中；

While($Cuboid_{i-1}$ 中各个维的维层次不为最粗粒度)

/*递归地对生成的 Cube 进行分割和分组聚集计算，直至最粗粒度的层次为止*/

CreateDHAC（DHAC $Cuboid_{i-1}$); }}}

例如以一个二维稀疏 cube 为例，它的两个维分别为 DimTime{Day,Month,Year} 和 DimRegion{City,Province,Country}，其部分数据如图 4-11 所示，所创建的 DHAC 如图 4-12 所示。对这个稀疏 Cube 中的数据在 DHAC 中采用线性的 *Chunk* 数据块进行存储，有利于压缩存储空间。

MBB11 3 1 2 2 4	3 MBB12 4 6 2	3 MBB21 4 4	MBB22 6 2 1 5
MBB13 7 3 5	3 MBB14 3 4	6 MBB23 5	6 MBB24 2 3
MBB31 5 2 3	MBB32 2 7	7 MBB41 1 5	6 MBB42 5
2MBB33 4 5	3 MBB34 4 2 1	MBB43 2 8	8 MBB44 6

图 4-11 二维稀疏 Cube

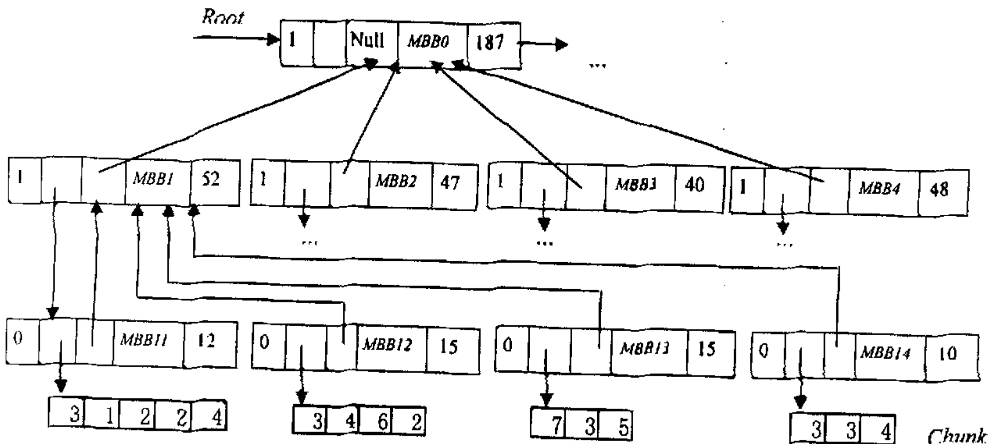


图 4-12 维层次聚集 Cube（DHAC）

为了提高 Cube 中数据的存取速度和分组聚集速度，可以将维层次聚集 Cube 中的各数据块

Chunk 可以按 Cube 中结点的层次次序和各个维相应顺序进行簇集存储。对维层次聚集 Cube 中的范围 *MBB* 采用维层次编码方式来表示，按维层次编码组合的混合编码的 Hash 函数来计算其存储地址，作为 Cube 中数据单元的存取依据。

4.4.2 DHAC 的快速查询算法

定义 4.7（MOLAP 查询重写）．在维层次聚集 Cube 中，可以将 MOLAP 查询 Q 定义为 Q（*SG*，*MBB*，*AG*，*AGG*）。

其中：$SG=(S_1,S_2,\cdots,S_d)$ 为查询 Q 的选择粒度，S_i 为维层次 DH_i 中的维层次粒度；$MBB=\{MBB_i\}$ 是查询 Q 的选择范围，$MBB_i=(I_{i1}, I_{i2},\cdots,I_{id})$是由查询 Q 选择谓词所决定的 d 个维的维层次属性值范围，I_{ij} 表示维层次 DH_i 中第 j 个维层次属性值范围：$AG=(A_1,A_2,\cdots,A_d)$ 是查询 Q 的聚集粒度，A_i 为维层次 DH_i 中的维层次粒度；$AGG=\{agg(m)|agg\in\{MIN,MAX,SUM,COUNT\},m$ 是度量属性$\}$为聚集值。

假设用户提交的查询 Q 如下：

```
SELECT city,month,SUM(SaleNum)
FROM Sales,DimRegion,DimTime
WHERE Sales.RegionID=DimRegion.RegionID AND Sales.TimeID=DimTime.TimeID
      AND (DimRegion.city='南京' OR DimRegion.city='苏州')
      AND DimTime.month≥2001.2  AND DimTime.month≤2001.3
GROUP BY city,day
```

这个查询 Q 可用 Q(*SG*,*MBB*,*AG*,*AGG*)=Q((city,month,all),{(['南京','南京'],[2001.2,2001.3],(-∞,+∞)),(['苏州','苏州'],[2001.2,2001.3],(-∞,+∞))},(city,day,all),{SUM(SaleNum)})来描述。

定义 4.8（查询 Q 与 Cube 结点包含关系）．设查询 Q 的查询范围为 $MBB_q=\{(qx_i,qy_i), 1\leqslant i\leqslant d\}$，DHAC 中各结点所包含的范围为 $MBB_c=\{(cx_i,cy_i), 1\leqslant i\leqslant d\}$，则它们之间的包含关系为：

（1）如果 $MBB_c\subseteq MBB_q$（即 $cx_i\geqslant qx_i$ AND $cy_i\leqslant qy_i$），则 MBB_q 包含 MBB_c。

（2）如果 $MBB_c\cap MBB_q=\Phi$（即 $cx_i>qy_i$ OR $cy_i<qx_i$），则 MBB_q 与 MBB_c 不相交。

（3）如果 $MBB_c\cap MBB_q\neq\Phi$（包括 $MBB_q\subseteq MBB_c$），则 MBB_q 与 MBB_c 相交。

MOLAP 查询 Q 与 DHAC 的包含关系如图 4-13 所示。

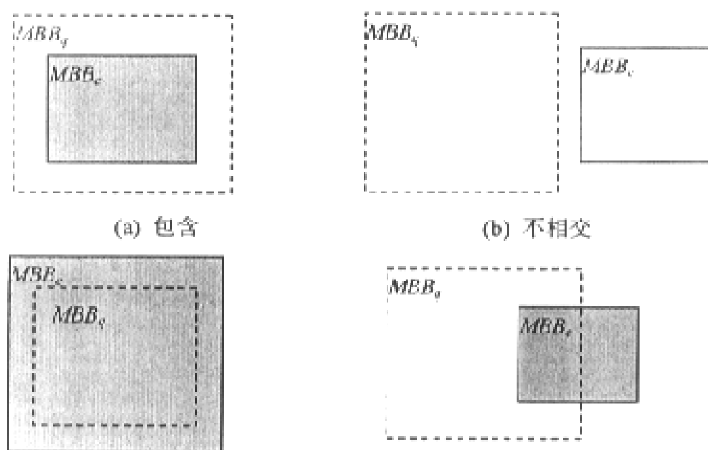


图 4-13 查询 Q 与 DHAC 的包含关系

当用户向数据仓库提交 MOLAP 查询 Q 时，MOLAP 服务器中的查询 Agent 根据用户提交的 MOLAP 查询 Q 和元数据库中 Cube 元数据，选择相应决策主题的 DHAC。根据 MOLAP 查询 Q 中各个维属性值范围，确定 Q 的查询范围 MBB_q，按深度优先方式从这个 DHAC 的根结点开始，选取相应结点与其 MBB_c 进行比较，按以下三种情况进行迭代计算查询变量值 Q_{answer}，直至查询 Q 的范围 MBB_q 与 DHAC 中某一层 Cube 的所有结点 MBB_c 都不相交时为止：

（1）如果包含，则将查询 Q 的查询变量值 Q_{answer} 与 MBB_c 表示的 Cube 中结点的 *AGG* 进行聚集计算，则不必对其所覆盖的下层结点进行访问，只对其包含结点的聚集值进行聚集计算。

（2）如果相交，则沿着 *pointer* 指针依次选择下一层次结点的 MBB_c 进行比较。

（3）如果不相交，则修剪该分支，对其所覆盖的下面结点不必进行访问，只须与下一个结点的 MBB_c 进行比较。

则计算出查询变量值 Q_{answer} 即为用户所提交的查询结果。这样，只须对相交的结点进行递归地比较它们的 *MBB* 范围，对包含的结点进行聚集计算，从而大大减少对 Cube 结点及存储聚集值的 *Chunk* 的访问次数，提高了其查询效率。其查询算法形式描述见算法 4.5。

```
算法 4.5：RangeSum( Q,DHAC_Cube Cube)
{ MOLAP 中的 Agent 根据用户提交的查询 Q 选择相应决策主题的 Cube;
  int AGG=0;
  do { if (Cube.DHAC_TreeNode_i.MBB_c ⊆ Q.MBB_q)          /*在查询 Q 与 Cube 包含时*/
          Q_answer + = Cube.DHAC_TreeNode_i.Cell.AGG;      /*对于 SUM*/
       else if(Cube.DHAC_TreeNode_i.MBB_c ∩ Q.MBB_q ≠ Φ) /*在 Q 与 Cube 相交时*/
          Q_answer+=RangeSum( Q, Cube.DHAC_TreeNode_i.ptrChild);
       else                                                  /*在 Q 与 Cube 不相交时*/
          Q_answer+=RangeSum( Q, Cube.DHAC_TreeNode_i.ptrSucc);
     }until (与某一层 Cube 中结点 MBB 都不相交时)
}
```

4.4.3 DHAC 的增量更新算法

1) 数据更新时的增量更新算法

Cube 中的数据更新主要分为对现有的数据单元中的数据进行修改、插入新的数据单元和删除数据单元等情况。在维层次聚集 Cube 中的数据单元发生数据更新时，只须根据维层次聚集树从维层次聚集 Cube 根结点开始由上向下将 Cube 中各结点的范围 MBB_c 与更新数据单元所在的 *MBB* 依次进行比较，找到覆盖更新数据单元所在 *MBB* 的叶结点 *Chunk*，从该叶结点所指向的数据 *Chunk* 用 *address* 选取相应的更新单元 *cell*，分别按下列不同情况对更新单元 *cell* 进行数据更新：

（1）如果是数据修改，则用 *updateAGG*（对于 SUM 来说，*updateAGG = newAGG-oldAGG*）对更新单元 *cell* 进行数据更新，其子函数为 *UpdateCell*（）。

（2）如果是插入数据，则在查找到更新结点所指向的 *Chunk* 中插入新数据的数据单元，将 {*address,addAGG*} 赋给该数据单元，其子函数为 *addCell*（）。

（3）如果是删除数据，则将删除数据的数据单元 *cell* 从数据 *Chunk* 中去除（可以作一个删除标记，必要时进行批量回收空间），其子函数为 *deleteCell*（）。

然后再从更新的数据单元沿 *parentPtr* 指针向上对所有受到其更新影响的祖先结点的相应数据单元进行增量更新，其子函数为 *updateParentCell*（）。

假设 *newAGG*、*oldAGG* 分别为单元更新前、更新后的聚集值，*updateAGG* 为更新前后的差值，*addAGG* 为插入新单元的聚集值，*cell* 为 *Chunk* 中更新的数据单元。

DHAC 中数据更新时的增量更新算法形式描述见算法 4.6。

算法 4.6:

```
/*数据更新子函数 UpdateCell（）*/
UpdateCell(DHAC Cuboid_i, int newAGG)
  {  根据维层次聚集树从 DHAC 的根结点开始以由上向下方式查找到这个更新数据所在的叶
     结点 DHAC_TreeNode_i 的相应数据 Chunk 中的更新单元 Cell;
     oldAGG = DHAC_TreeNode_i.Chunk.Cell.AGG;
     DHAC_TreeNode_i.Chunk.Cell.AGG= newAGG;
     updateAGG= newAGG- oldAGG;              /*SUM 为例，计算更新前后的差值*/
     UpdateParentCell(DHAC Cuboid_i, int updateAGG);
     /*对所有受到其更新影响的祖先结点进行增量更新*/
  }

/*数据插入子函数 addCell（）*/
addCell(DHAC Cuboid_i, int addAGG)
  {  根据维层次聚集树从 DHAC 根结点开始以由上向下方式查找到这个插入数据单元所在的
     叶结点 DHAC_TreeNode_i 及其插入数据单元所在的数据 Chunk;
     在这个插入数据单元所在的数据 Chunk 中申请一个新的数据单元 cell，计算其偏址 address，
     将{address,addAGG}赋给该 cell 单元;
     updateAGG= addAGG;
     UpdateCell(DHAC Cuboid_i, int updateAGG);
     /*对所有受到插入数据单元更新影响的祖先结点进行增量更新*/
  }

/*数据删除子函数 UpdateCell（）*/
deleteCell(DHAC Cuboid_i, DHAC_TreeNode_i.Chunk.Cell)
 { 根据维层次聚集树从 DHAC 根结点开始以由上向下方式查找到这个删除数据所在的叶结点
   DHAC_TreeNode_i 的相应数据 Chunk 中的更新单元 cell;
   将这个数据单元 cell 从 DHAC_TreeNode_i.Chunk 去除或作个删除标记;
   updateAGG= - DHAC_TreeNode_i.Chunk.Cell.AGG;
   /*用更新单元聚集值 DHAC_TreeNode_i.Chunk.Cell.AGG 的反值对其祖先结点进行增量更新*/
   UpdateParentCell(DHAC Cuboid_i, int updateAGG);
}

/*对所有受到更新单元数据更新影响的祖先结点进行增量更新的子函数 UpdateParentCell（）*/
UpdateParentCell(DHAC Cuboid_i, int updateAGG)
{while (not Root or 祖先结点 DHAC_TreeNode_j 的 Chunk 中 Cell.updateAGG≠0)
  { DHAC_TreeNode_i= DHAC_TreeNode_i->parentPtr;
    oldAGG = DHAC_TreeNode_i.Chunk.Cell.AGG;
    for (所有更新的孩子结点)
        {DHAC_TreeNode_i.Chunk.Cell.AGG+= updateAGG;
         updateAGG= DHAC_TreeNode_i.Chunk.Cell.AGG- oldAGG;}
    return updateAGG;
  }}/*沿 parentPtr 指针向上，对其所有受到更新影响的祖先结点 Chunk 中的相应数据单元 Cell，
  用 updateAGG 进行增量更新，直到某个祖先结点 AGG 值不再更新时或根结点为止*/
```

如在图 4-10 所示的 DHAC 中，将<2001 年 1 月,南京>的销售值从 6 修改到 8（即图 4-10 中圆圈内标志的数据单元），则用修改前后的差值 *updateAGG*=2 沿着 DHAC 由下向上依次加到<2001 年 1 季度,江苏>、<2001 年,华东地区>数据单元 *cell* 的聚集值上（如图 4-10 中圆圈内标志的数据单元），来对 DHAC 进行增量更新。

2) 增加新维时的增量更新算法

在插入新维及其维成员数据单元时，以往 Cube 是不能进行增量更新，只得重新构建 Cube。但在维层次聚集 Cube 中，不需要重新构建 Cube，只需根据插入新维的维层次树，沿着维层次聚集树由下向上对维层次聚集 Cube 中所有子立方体 *Cuboid* 的范围 $MBB_c=\{c_1,c_2,\cdots c_d\}$ 中添加一个元素 c_{d+1}，对发生数据更新的数据单元聚集值 *AGG* 递归地进行更新，直到根结点，来实现维层次聚集 Cube 在插入新维时的模式增量更新。其算法形式描述见算法 4.7。

```
算法 4.7：DHAC AddDim (DHAC Cuboid_i, MBB_i )
  { if(Cuboid_i 中各个维的维层次不是最粗粒度)
      for(j=0;j<M;j++)
        {向聚集Cube 中各叶结点 Cuboid_ij 子立方体的聚集范围 MBB_i 中添加新插入维的聚集范围
         c_{d+1};
         addCell(DHAC Cuboid_i, int addAGG); }
      while (Cuboid_{i-1} 中各个维的维层次不是最粗粒度)
        { Cuboid_{i-1}.TreeNode_i= Cuboid_i.TreeNode_i->parentPtr;
          AddDim (DHAC Cuboid_{i-1}, MBB_{i-1}); }
          /*沿着每一个子立方体 Cuboid_i 的 parentPtr 指针对上一层 Cuboid_{i-1} 进行增量更新*/ }
```

4.4.4 Cube 语义操作

在 DHAC 的存储结构中，利用维层次 B^+树中所保存的层次信息，可以进行上探、下钻等 Cube 语义操作。上钻时，就相当于数据沿着维层次 B^+树往上聚集。在 DHAC 中沿 *parentPtr* 指针向上对所对应的逻辑块进行聚集计算。在下钻时，在 DHAC 中沿 *childPtr* 指针向下对所对应的逻辑块进行访问，即相当于是一系列的范围聚集查询。

例如在上述的 DHAC 中，从<月,市>层次聚集 Cube 到<季度,省>层次聚集 Cube 进行的聚集查询过程就是 Cube 上探操作，可以沿<月,市>层次聚集 Cube 中相应结点的 *parentPtr* 指针向上对所对应的<季度,省>层次聚集 Cube 中的逻辑块进行访问。从<年,地区>层次聚集 Cube 到<季度,省>层次聚集 Cube 进行的聚集查询过程就是 Cube 下钻操作，可以沿<年,地区>层次聚集 Cube 中相应结点的 *childPtr* 指针向下对所对应的<季度,省>层次聚集 Cube 中的逻辑块进行访问。

4.5 性能分析

假设 DHAC 有 d 个维，DHAC 的层次为 h，在最明细的叶子 Cube 中各个维上共有 n 个不同单元。DHAC 的叶子 Cube 存储的是最明细数据，即层次粒度最细的数据，将这些明细数据按各个维的维层次进行分割，依次向上进行聚集生成父 Cube，则 DHAC 中每个维的各个层次的单元数依次为 $n^{\frac{h-1}{h}},n^{\frac{h-2}{h}},\cdots n^{\frac{2}{h}},n^{\frac{1}{h}}$ 个。在 DHAC 中查询，只需从根结点向叶结点进行 *MBB* 比较查询，即从根结点到叶结点的路径。在最佳情况下，只需要访问 DHAC 中每一个聚集结点，则最佳情况下其查询的时间复杂度为 O(1)。在最坏情况下，需要依次从 DHAC 的根结点到叶结点对与查询范围 *MBB* 相交的结点 *Cuboid* 进行遍历，需要遍历 h 次，每次遍历访问 $\log_h^d n$ 个数据单元。于是，在最坏情况下，

DHAC 聚集查询的复杂度为 $O(h\log_h^d n)$。

在 DHAC 的增量更新算法中，对于聚集 Cube 叶结点 *Cuboid* 中的某些数据单元值进行更新，只需对沿着更新单元所在 *Cuboid* 的 *parentPtr* 的指针由下向上，对所有层次粒度较粗且受到更新影响的父立方体 *Cuboid* 中的数据单元进行增量更新，于是，DHAC 增量更新的复杂度为 $O(\log_h^d n)$。

在 DHAC 与 SDDC、HDC 等其它传统聚集 Cube 上进行查询和更新操作的复杂度如表 4.1 所示。

表 4.1 各种 Cube 方法性能对比

聚集 Cube	查询	更新
PS Cube	$O(1)$	$O(n^d)$
RPS Cube	$O(1)$	$O(n^{d/2})$
DDC	$O(2^d log_2{}^d n)$	$O(log_2{}^d n)$
HDC	$O(log_2{}^d n)$	$O(log_2{}^d n)$
SDDC	$O(log_2{}^d n)$	$O(log_2{}^d n)$
DHAC	$O(h log_h{}^d n)$	$O(log_h{}^d n)$

由表 4.1 的复杂度分析可以得出，传统的聚集 Cube 相比，DHAC 具有最优的综合性能。

因为 SDDC、HDC 是目前对 PS 改进最好的存储结构，所以我们从聚集查询和增量更新代价来比较 DHAC 与 SDDC 的优劣。影响聚集 Cube 性能的主要因素有：Cube 中的维个数、Cube 中的数据量、Cube 中各个维的层次数。

第 1 组实验测试是聚集 Cube 中不同维个数和数据量对查询性能的影响。假设维个数分别为 3、4、5，维的层次数为 4，SDDC 采用二分法。测试数据的大小分别为Ⅰ：64×64×64，Ⅱ：128×128×128，Ⅲ：512×512×512；Ⅳ：64×64×64×64，Ⅴ：128×128×128×128，Ⅵ：512×512×512×512；Ⅶ：64×64×64×64×64，Ⅷ：128×128×128×128×128，Ⅸ：256×256×256×256×256。实验结果如图 4-14~4-16 所示。

第 2 组实验测试是聚集 Cube 中不同维个数和数据量对更新性能的影响。测试数据与第 1 组实验数据相同，实验结果如图 4-17~4-19 所示。

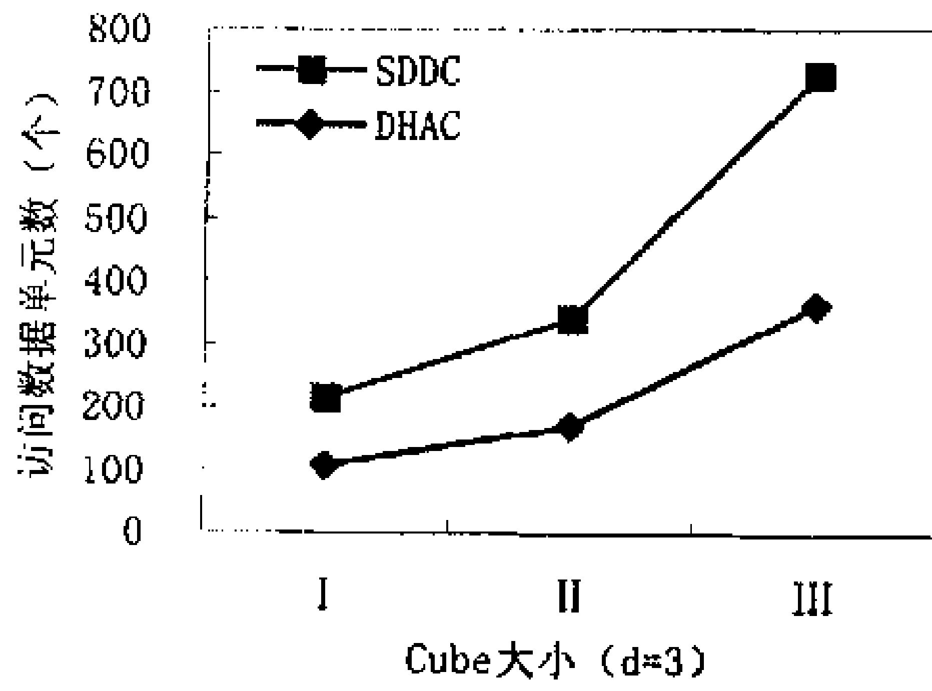


图 4-14 Cube 大小对查询影响（d=3）

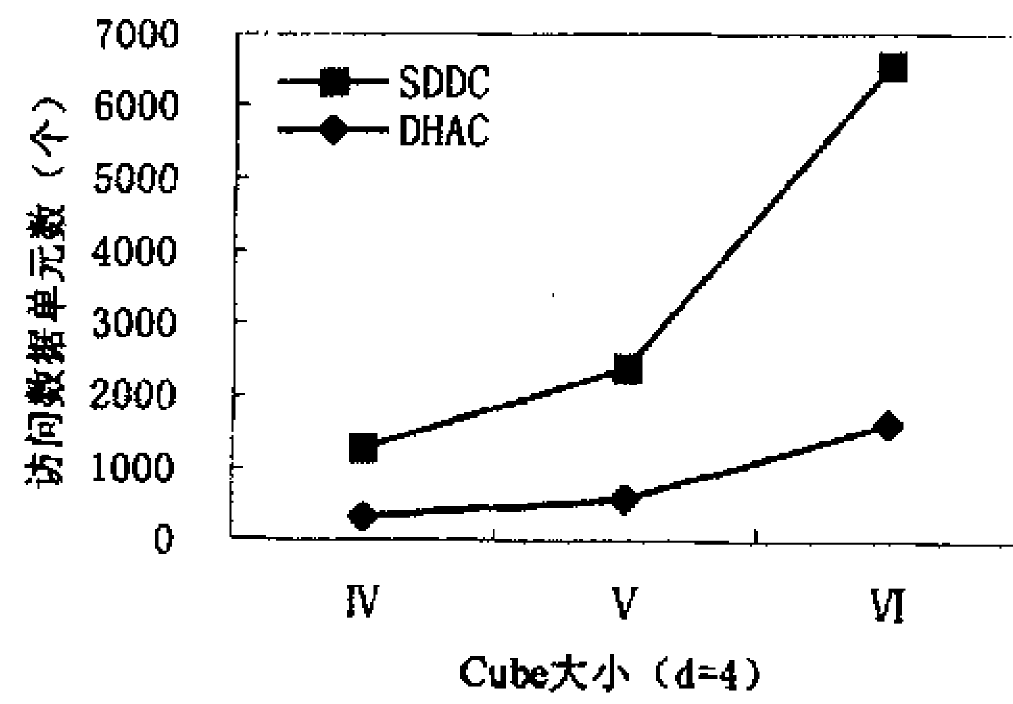


图 4-15 Cube 大小对查询影响（d=4）

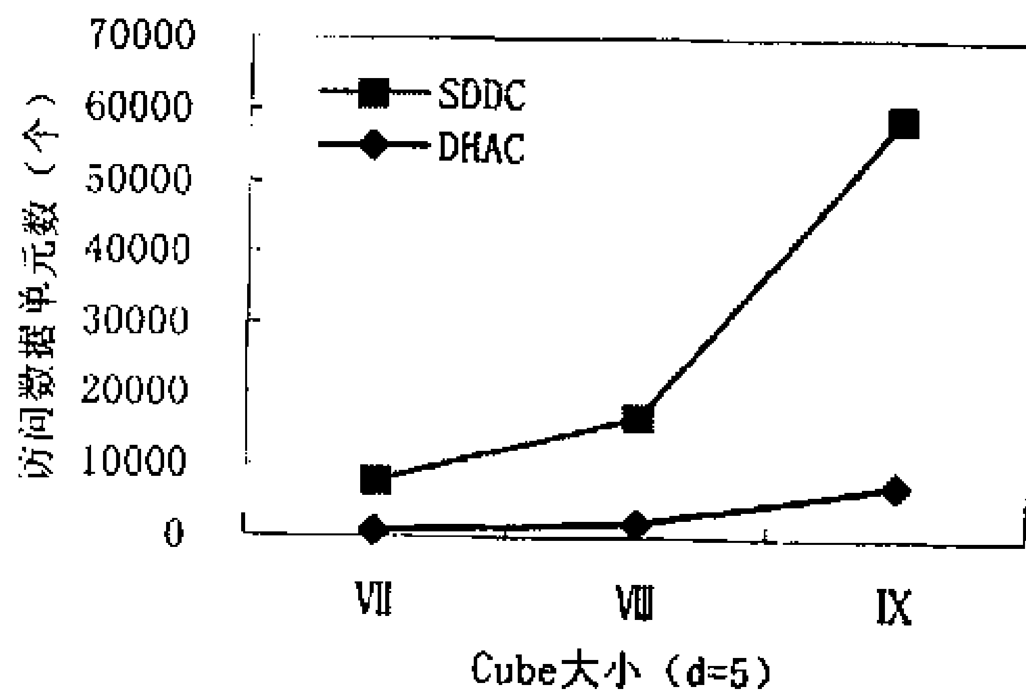


图 4-16 Cube 大小对查询影响（d=5）

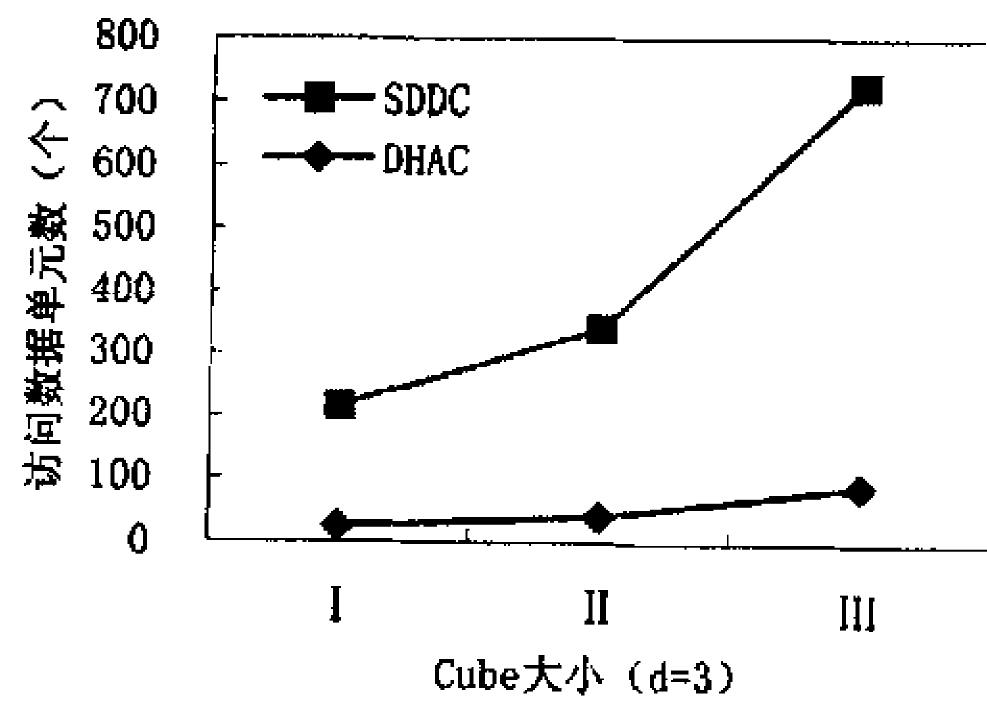


图 4-17 Cube 大小对更新影响（d=3）

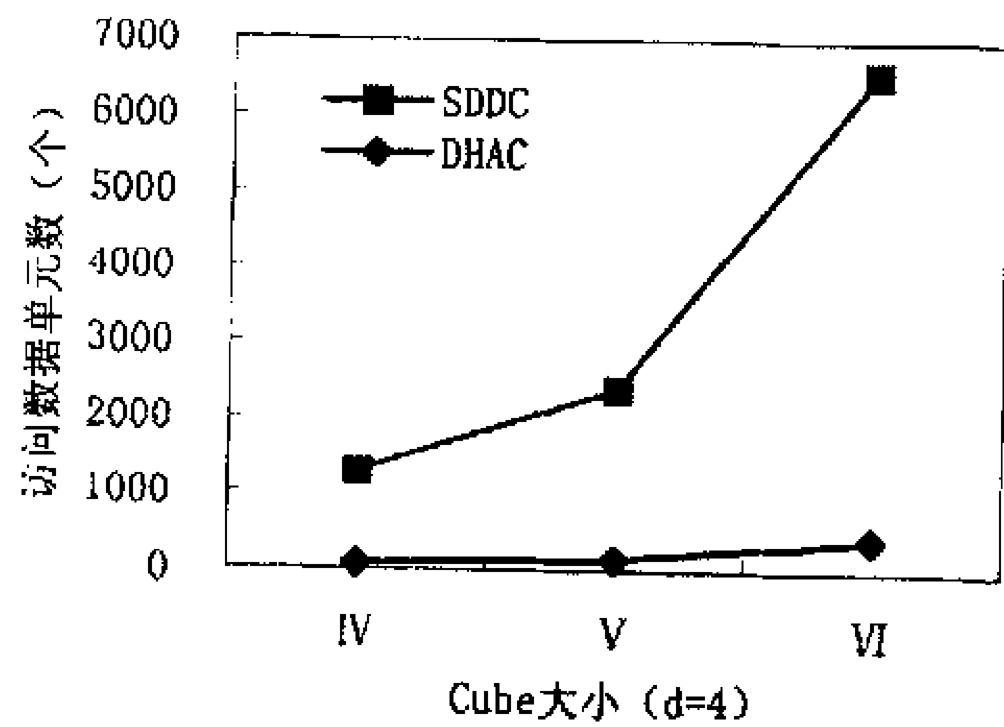


图 4-18 Cube 大小对更新影响（d=4）

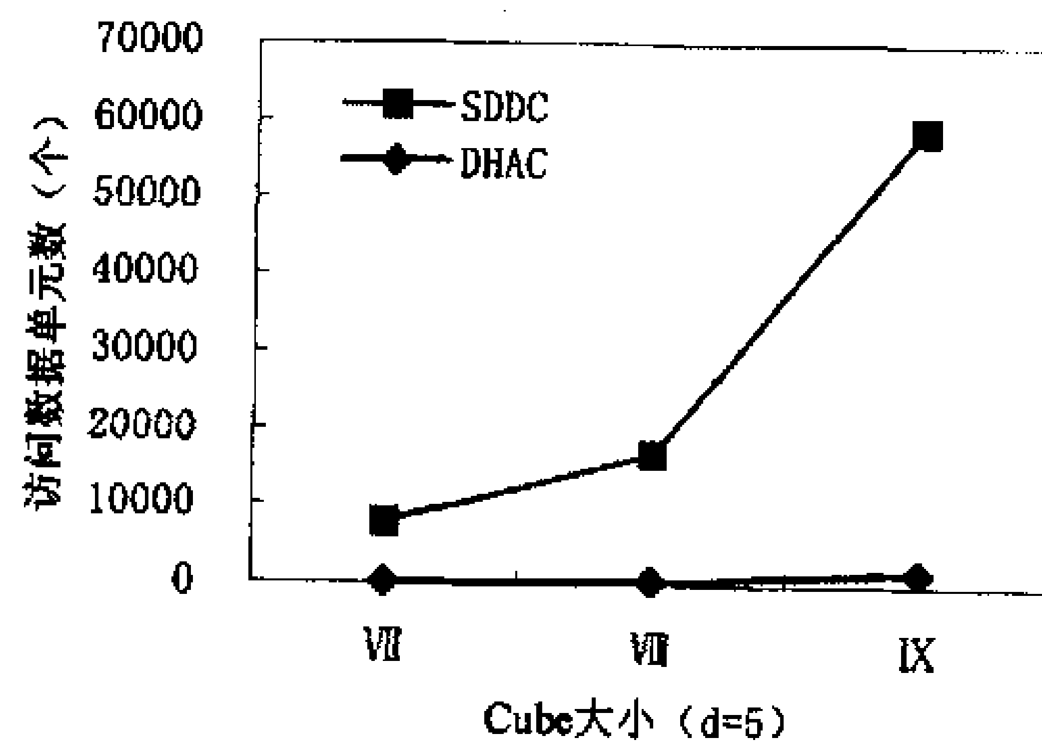


图 4-19 Cube 大小对更新影响（d=5）

可见，在维数相同时，随着数据量的增长，DHAC 与 SDDC 变化的趋势基本一致，但 DHAC 比 SDDC 访问的单元个数要少，这种差别随着维数的增加越来越明显。无论是哪种规模的数据，在维数较小时，对查询和更新性能的影响都不是很大，但在维数大的时候，SDDC 访问的单元个数增加幅度较大，而 DHAC 访问单元个数的变化较为平缓。

聚集 Cube 的综合性能 $CUBE_{cost}$ 可以通过式 4.9 来评价。

$$CUBE_{cost} = k_q Q_{cost} + k_u U_{cost} \tag{4.9}$$

式中：k_q、k_u 分别为数据查询和数据更新次数；

Q_{cost}、U_{cost} 分别为数据查询和数据更新代价。

第 3 组实验测试是通过 Cube 中维层次数的变化来对 DHAC 与 SDDC 的综合性能进行对比分析。其实验结果如图 4-20 所示。

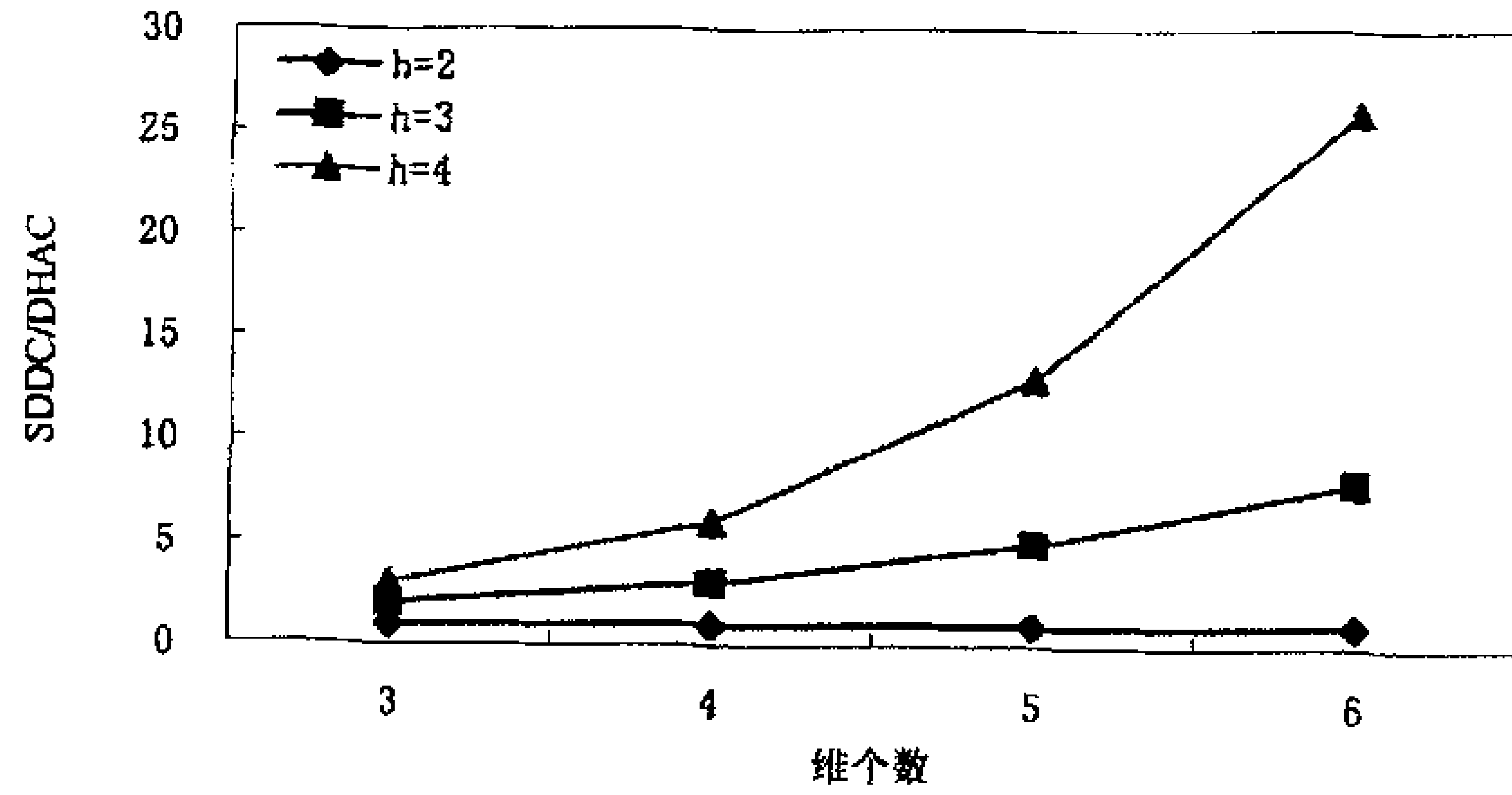


图 4-20 DHAC 与 SDDC 性能对比

对于 SDDC、HDC，因为查询和更新与维层次数无关。从图 4-20 可以看出，在层次数 h=2 时，DHAC 与 SDDC 的性能相差不大，但层次数 $h \geq 3$ 时，DHAC 的性能优于 SDDC 等，并且随着维个数及数据量的增加，这种性能优化就愈加明显。随着维层次数的增加，DHAC 的性能优化要比 SDDC 等更加明显。尤其是在维度较高，维层次较多，维的基数也较大时，DHAC 的综合性能要明显优于 SDDC 等。

4.6 本章小结

DHAC 是充分利用维层次聚集技术来创建的层次聚集 Cube，在用户提交 MOLAP 查询 Q 时，根据查询 Q 的查询空间 MBB_q 与 DHAC 中各结点的 MBB_c 进行比较，来计算 Q 的查询结果，从而提高了 MOLAP 查询效率。在 DHAC 中进行数据插入和删除等数据更新时，利用维层次聚集树中的维层次前缀，沿着维层次聚集树由下向上用更新前后的差值对更新单元所在结点的所有受更新影响的祖先结点进行增量更新，直到某一祖先结点不再更新为止，最差情况下更新到根结点。从而实现了对 DHAC 进行增量更新，提高了 Cube 的更新效率。在插入新的维及其数据时，只需对 DHAC 的范围 $MBB_c=\{c_1,c_2,\cdots c_d\}$ 中添加一个元素 c_{d+1} 即可，从而避免了以往 Cube 在插入新维时重构 Cube 所造成大量的时间开销，实现了不需要重新构建聚集 Cube 就可以对 DHAC 进行模式增量更新。由于 DHAC 保存了层次信息，能支持上探、下钻等层次语义操作。对 DHAC 与 SDDC、HDC 等聚集 Cube 进行了性能分析和比较，结果表明本文所提出的 DHAC 性能最佳。

第五章　实视图选择与分割及增量更新维护

在数据仓库中为了加快对大量数据的查询处理速度，通常将存储在基表中的原始数据进行聚集计算，并以实视图方式进行存储。从而造成在数据仓库中实视图可能达到几千个，这给查询和更新维护带来了很大的难度。因而如何选择数据仓库视图进行有效地分割和实体化、如何对实视图进行增量更新以及如何提高视图维护效率也成了 OLAP 的一个关键问题，它将直接影响 OLAP 查询效率和数据质量。

本章首先提出根据 OLAP 查询中的选择谓词来构造其最小项谓词，选择数据仓库立方体视图进行水平分割和实体化，给出了视图最佳选择与分割算法；然后提出利用视图计算的中间结果创建辅助视图，并在数据仓库中进行实体化，充分利用视图表达式树及其有效的增量维护算法来计算实视图和辅助视图的精确变化；给出了实现数据仓库视图增量更新维护的体系结构；最后是本章内容的总结。

5.1　实视图选择与分割

5.1.1　概述

在数据仓库中，OLAP 查询复杂，实视图数量多，如何有效地选择数据仓库视图进行分割和实体化，成为提高 OLAP 查询效率中的一个重要问题。本章节提出通过从 OLAP 查询穷举的简单选择谓词构造的最小项谓词来选择最佳数据立方体视图进行水平分割。通过数据仓库中数据立方体视图的贪婪效益值来选择最佳视图进行分割，要比直接对事实表[84,86,87]进行分割效率要高，因为前一种方法让用户可以根据决策主题和查询访问频率来选择最佳视图及其相应裂片来回答查询，减少被查询访问的视图元组数，从而减少查询响应时间。

定义 5.1 设 A 为一个命题，对 A 做所有可能的解释 I，对于这些解释 I，若 $I(A)$皆为真，则称 A 为永真式；若 $I(A)$皆为假，则称 A 为永假式。在含有 m 个谓词的简单合取式中，若每个谓词与其否定不同时存在，而两者之一出现一次且仅出现一次，则称该简单合取式为最小项谓词。

例如两个谓词 P_1 和 P_2，它们构成的最小项谓词是 $P_1 \wedge P_2$，$P_1 \wedge \neg P_2$，$\neg P_1 \wedge P_2$ 和 $\neg P_1 \wedge \neg P_2$。

引理 5.1 一个谓词公式 A 为永假式的充要条件是 A 的析取范式中每个简单合取式至少包含一个谓词及其否定；为永真式的充要条件是 A 的合取范式中每个简单析取式至少包含一个谓词及其否定。

证明：参见文献[102]，这里省略。

引理 5.2 由最小项谓词的真值表和引理 5.1 可以得出，对于有 m 个谓词构成 2^m 个最小项谓词 M_i（$1 \leq i \leq 2^m$），具有以下性质：

（1）　任意两个不同的最小项谓词的合取式是永假式，即 $M_i \wedge M_j \Leftrightarrow$ F（$i \neq j$）。

（2）　所有最小项谓词的析取式为永真式，即 $\bigvee_{i=0}^{2^m} M_j \Leftrightarrow$ T。

定义 5.2 谓词的重要效益值（IP）用来表示其被访问的重要程度，是由含有该谓词的查询访问频率与谓词的基数(即在视图中该谓词值为真的元组数)乘积之和。

定义 5.3 视图的贪婪效益值用来表示视图被选择的优先级。视图的贪婪效益值可以按式 5.1 来计算。

$$\text{视图的贪婪效益值} = \sum_{q_i | access(q_i, v)=1} \frac{\sum_{f_m | access(q_i, f_m)=1} (|f_m| * AF_{q_i})}{\sum AF_{q_i}} \tag{5.1}$$

式中：$|f_m|$ 是表示通过 f_m 分割的最小项谓词确定的元组数；

AF_{q_i} 是表示查询 q_i 对裂片 f_m 的访问频率；

$access(q_i, f_m)=1$ 表示查询 q_i 访问了裂片 f_m，$access(q_i, v)=1$ 表示查询 q_i 访问了视图 v。

根据 OLAP 查询穷举出其简单选择谓词，这些简单选择谓词可以定义为"PA_{ij} θ value"，其中 θ $\in\{<,>,\neq,\leq,\geq,=\}$，$PA_{ij}$ 表示第 i 个查询中的 where 子句中第 j 个选择谓词中的属性，value $\in$ Dom(PA_{ij})。根据查询的访问频率计算谓词的重要效益值（IP），选择 p 个 IP 值最大的谓词。用这些访问视图的简单选择谓词来构造其最小项谓词，对最小项谓词进行优化，利用优化过的最小项谓词作为该视图水平分割的依据。

定义 5.4 访问视图 V_i 的用户查询可以分解成分析属性 AA、分段属性 PA 和度量属性 MA 三个属性。其中 AA 是查询 Q_i 中的分组属性，PA 是查询 Q_i 中的 WHERE 子句中的选择属性；MA 是查询 Q_i 中的聚集属性。

定理 5.1. 由视图上的查询选择谓词所构成的最小项谓词，对该视图进行水平分割，满足关系分割的 3 条准则[1]，即：

（1）完备性：全局关系的所有数据都要分配到相应的裂片中，否则，将会因分割而丢失数据。

（2）不相交性：在水平分割的裂片中，应该没有重复的元组。

（3）可重构性：可以由裂片重构全局关系。

证明：可以用以下的 SQL 语句描述在水平分割后的视图上进行的查询：

```
SELECT * FROM V WHERE M;
```

其中：V 为视图，M 是由 p 个所选择的简单选择谓词构成的最小项谓词集，用于对视图进行水平分割；设 V 被分成的 m 个裂片为 V_1，V_2，…，V_m，对应这 m 个裂片的最小项谓词分别为 M_1，M_2，…，M_m。

（1）证明完备性

由引理 5.1 和引理 5.2 可得 $M_1 \vee M_2 \vee \cdots \vee M_m = T$，即{SELECT * FROM V WHERE ($M_1 \vee M_2 \vee \cdots \vee M_m$);} $=V$，得出 $V_1 \cup V_2 \cup \cdots \cup V_m = V$，证得这种分割满足关系分割的完备性准则。

（2）证明不相交性

由引理 5.1 和引理 5.2 可得 $M_i \wedge M_j = F$（$i \neq j$），即 {SELECT * FROM V WHERE ($M_i \wedge M_j$);} $=\Phi$，得出 $V_i \cap V_j = \Phi$（$i \neq j$），证得这种分割满足关系分割的不相交性准则。

（3）证明可重构性

将视图分割的所有裂片进行"并"操作就可以实现视图的重构，即：

```
SELECT * FROM V1 WHERE M1
UNION SELECT * FROM V2 WHERE M2
……
UNION SELECT * FROM Vm WHERE Mm;
```

从而证得按这 p 个谓词构成的最小项谓词进行的视图分割满足关系分割的三个准则，即这些裂片与原视图是等价的。

5.1.2 实视图选择与分割算法

图 5-1 为实视图选择与分割的整体结构。通过视图分割算法从 OLAP 查询中穷举其选择谓词，构造其最小项谓词，从原数据仓库中选择最佳立方体视图进行水平分割。用元数据的形式将视图和分割的裂片定义保存到元数据库中，相应数据保存至水平分割的 DW 或数据集市中。为了有效地、快速地回答用户提交的 OLAP 查询，在决策者等用户 OLAP 查询访问工具与存储水平分割视图的数据仓库之间，设计一个基于裂片的查询访问组件，来预先确定回答查询的视图及其相应裂片。从而减少被查询访问的视图元组数，减少查询响应时间，快速响应用户提交的 OLAP 查询。

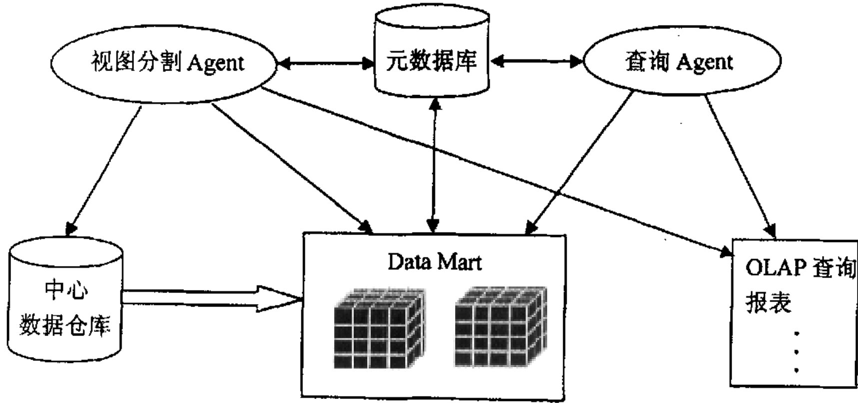


图 5-1 实视图选择与分割的整体结构

视图选择与分割算法是对贪婪算法进行改进后的算法。利用改进的贪婪算法，迭代地从数据仓库立方体网格中，选择最佳的立方体视图。根据用户访问的 OLAP 查询穷举其选择谓词，选择部分重要谓词来构造其最小项谓词，用这些最小项谓词对选择的视图进行水平分割。同时重新计算分割视图的贪婪效益值，选择下一个最佳视图进行水平分割。将水平分割视图和分割后的裂片定义以元数据的形式保存到元数据库中。实视图与裂片的元数据模式如图 5-2 和图 5-3 所示。视图选择与分割主要算法如算法 5.1 所示。

View_Id	视图名称	类型	存储位置	选择谓词	连接谓词	显示属性	基表

图 5-2 实视图元数据模式

Fragment_Id	裂片名称	View_Id	存储位置	最小项谓词	最小项谓词组成

图 5-3 裂片元数据模式

算法 5.1：实视图选择与分割算法

{ /*输入：选择视图个数 n，带有视图贪婪效益值的立方体网格，查询集 $\{Q_m\}$，查询访问频率 $\{AF_m\}$；输出：n 个选择视图 V_i 的裂片集 $\{V_i^j\}$ */

$SetV$ = {最佳视图}；

V_i =最佳视图;

for(i=1;i<=n;i++)

{ 谓词集 P_i={};

谓词的 IP 初始化为 0;

for(m = 1;m<=查询的个数;m++)

{ 从查询 Q_m 生成选择谓词 P_m;

$P_i = P_i \cup P_m$;

for each $P_m^j \in P_m$

P_m^j 的 $IP = P_m^j$ 的 $IP + (AF_m * \mid P_m^j \mid)$;

}

从 P_i 中选择 p 个 IP 值最大的谓词;

按照式 5.2 构造所选择的 p 个谓词的最小项谓词集 $\{m_i\}$；

进行最小项谓词集 $\{m_i\}$ 优化;

```
用元数据的形式将选择的 Vi 视图存储到元数据库中;
for each mj∈{mi}
      进行视图分割，并用元数据将最小项谓词 mj 及分割的裂片 Vi^j 定义存储
      到元数据库中，分割后的相应数据存储到水平分割的DW或数据集市中;
Query_total=0;
Frequency_total=0;
for each Qi∈{Qm}
   {for each Fi∈ {Vi^j}          /* Fi为分割的裂片*/
      {Query_total = Query_total + 在 Fi 上的|Qi| * AFi;
       if(在 Fi 上的|Qi|≠0)
            Frequency_total = Frequency_total + AFi;
      }
     视图 Vi 的新贪婪效益值= Query_total/Frequency_total;
   }
for each 子视图 Vj ∉SetV
      用已分割的父视图的贪婪效益值来计算视图 Vj 的贪婪效益值;
 SetV=SetV∪具有最大效益值的 Vj;
 Vi =Vj;
}}
```

算法5.1主要步骤如下：

（1）根据OLAP查询穷举出查询的简单选择谓词集$\{PA_{ij}\}$，确定查询Q_i的每个谓词P_{ik}的相应IP值。

（2）根据计算谓词IP值，选择p个IP值最大的谓词，即选择p个最重要的谓词集$P_i=\{P_{i1}, P_{i2},\cdots, P_{ip}\}$，按式5.2构造出这些谓词的最小项谓词集$M_i=\{M_{i1}, M_{i2},\cdots, M_{iz}\}$；

$$M_{ij} = \bigwedge_{P_{ip}\in P_i} P_{ip}^{*}\ ,\ 1\leqslant p\leqslant m\ ,\ 1\leqslant j\leqslant z \tag{5.2}$$

式中：$P_{ip}^{*}=P_{ip}$或$P_{ip}^{*}=\neg P_{ip}$。

（3）利用优化的最小项谓词对选择的最佳视图进行水平分割。用元数据将最小项谓词m_j及分割的裂片V_i^j定义存储到元数据库中，将分割后的相应数据保存到水平分割的DW或数据集市中。

（4）重新计算已进行水平分割的视图贪婪效益值，计算与分割视图有关的所有视图的贪婪效益值，选择下一个最佳视图进行水平分割，循环执行步骤（1）~（4），直至n个视图全部分割完毕。

利用设计在决策者等用户OLAP查询访问工具与存储水平分割视图的数据仓库之间的查询访问Agent，从用户提交的查询Q中分解出查询选择谓词。通过查询选择谓词从元数据库中查找相应的元数据，根据元数据定义从水平分割的DW或数据集市中选取相应的视图及其相应裂片，来回答用户提交的查询。从而减少查询所扫描的元组数和查询时间，提高查询效率，达到快速地、有效地回答用户提交的OLAP查询。

5.1.3　事例

为了便于说明，这里我们选用某企业销售数据仓库，其事实表和维表如下：

销售表（产品ID，公司ID，时间ID，销售量，销售额）

产品表（产品ID，类ID，大类ID，产品名，类名，大类名）

公司表（公司ID，市名，省名，国名）

时间表（时间ID，日，月，季，年）

假设P、S、T分别表示事实表中的产品、公司和时间三个维，其带有贪婪效益值的立方体网格如图5-4所示。

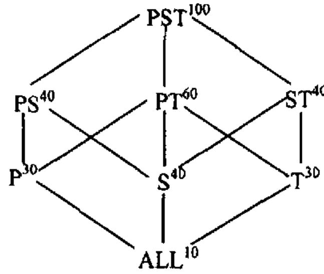


图5-4 立方体网格

产品ID的域是p0001，p0002，…，p1000。时间ID的值是用事务发生的年月日时分来表示。尽管这个表已上万条记录，为了便于说明，这里给出一个仅有10个元组的事实表数据样本来进行说明，如表5.1所示。

表5.1 数据仓库事实表数据样本

产品ID	公司ID	时间ID	销售量	销售额
P0001	S1	199907210703	2	20
P0100	S3	199907210840	500	4500
P1000	S4	199907211020	300	4800
P0001	S1	199907222140	5	50
P0100	S2	199907231000	3	30
P0010	S2	199907231000	4	60
P1000	S4	199907231640	100	1600
P0011	S1	199907232000	3	60
P0010	S4	199907240820	4	60
P0001	S3	199907241820	600	5400

假设对上述销售数据仓库进行一些OLAP查询如下：

Q_1：给出每个月在子公司S1的每种产品销售量，其属性为：*PA*=公司ID（*S*）；*AA*=月（*T*），产品ID（*P*）；*MA*=SUM（销售量）；谓词P_1：*S*="S1"。

Q_2：给出在上午已销售的产品销售量，其属性为：*PA*=时间ID（*T*）；*AA*=none；*MA*=SUM（销售量）；谓词P_2：hour（*T*）≤1200。

Q_3：查询每天在中午（即在12:00到13:00时间内）每个子公司的产品销售额，其属性为：*PA*=时间ID（*T*）；*AA*=公司ID（*S*）；*MA*=SUM（销售额）；谓词P_3：hour(*T*) >1200 and hour(*T*) ≤1300。

Q_4：查询在子公司S3中每分钟每种产品的销售额，其属性为：*PA*=公司ID（*S*）；*AA*=产品ID（*P*），时间ID（*T*）；*MA*= SUM（销售额）；谓词P_4：*S*="S3"。

假设查询Q_1~Q_4访问数据仓库的频率分别为：100,40,20和20次，从数据样本可以求得各简单谓词的基数（即由其确定的元组数）为：$|P_1|=3$，$|P_2|=6$，$|P_3|=0$，$|P_4|=2$。由上述查询属性可知，回答这些查询的可能视图有PST，T，ST。由图5.4的立方体网格可知，视图PST的贪婪效益值最大，则取视图PST。对于视图PST，计算出各谓词的*IP*值为：P_1的$IP=3\times100=300$，P_2的$IP=6\times40=240$，P_3的$IP=0\times20=0$，P_4的$IP=2\times20=40$。

根据上述的事例，如果*p*值为2，则*IP*值最大的谓词P_1和P_2将被选择。接着用被选择的P_1

和 P_2 简单选择谓词来构造的最小项谓词如下：

$M_1=P_1\wedge P_2$，　即 $S=$"S1"$\wedge$hour(T)$\leqslant$1200;

$M_2=\neg P_1\wedge P_2$，　即 $S\neq$"S1"$\wedge$hour(T)$\leqslant$1200;

$M_3=P_1\wedge\neg P_2$，　即 $S=$"S1"$\wedge$hour(T)$>$1200;

$M_4=\neg P_1\wedge\neg P_2$，　即 $S\neq$"S1"$\wedge$hour(T)$>$1200。

对于数据仓库立方体网格的顶层视图 PST，在 10 元组的样本表中我们会发现选择满足最小项谓词 M_1 的元组，只有产品 ID＝“P0001”一个元组，故 $|M_1|=1$，同理可求得 $|M_2|=5$，$|M_3|=2$，$|M_4|=2$。

视图 PST 被查询 Q_1~Q_4 访问的裂片、裂片的元组数、查询访问频率和元组总个数以及分割视图的新贪婪效益值如表 5.2 所示。

表 5.2　视图 PST 的裂片

查询	谓词	访问裂片	裂片的元组数	访问频率	元组总个数(裂片元组数×访问频率)
Q_1	P_1: $S=$"S1"	F_1, F_3	3	100	300
Q_2	P_2: hour(T) $\leqslant$1200	F_1, F_2	6	40	240
Q_3	P_3: hour(T) $>$1200 AND hour(T) $\leqslant$1300	F_3, F_4	4	20	80
Q_4	P_4: $S=$"S3"	F_2, F_4	7	20	140
合计				180	760
视图的新贪婪效益值＝元组总个数合计/访问频率合计＝760/180＝4.22　或　对上取整为 5					

如果当用户通过上述的 OLAP 查询 Q_1 来访问水平分割的数据仓库时，查询访问组件则确定由视图 PST 及其裂片 F_1 和 F_3 来回答查询 Q_1，用 SQL 语句表述如下：

```
SELECT  *  FROM  PST_F1  WHERE P1
UNION
SELECT  *  FROM  PST_F3  WHERE P1
```

显然，通过对这些裂片的直接访问就能满足查询要求，从而大大减少查询访问的平均元组数，改善查询响应时间。

5.2　实视图增量更新维护

5.2.1　概述

OLAP 中涉及大量从源数据中提取的总结数据，一般以实视图方式在数据仓库中存储，随着源数据的更新，这些实视图也需要进行相应的更新维护，以确保查询的数据质量和数据的准确性。因而，视图增量更新维护成为数据仓库的重要问题。由于数据仓库中的实视图很多，且数据更新会影响 OLAP 对用户的服务，每天可用于维护的时间是有限的。因此，提高对实视图的维护效率是 OLAP 中的一个关键问题[1,3,7]。

视图中数据更新一般采用从头重新生成和增量维护两种方法，重新生成需要更新的总结数据是很费时的，一般采用增量维护方法。所谓增量维护就是根据源数据的改变和视图中的旧值，推算出视图中的新值。源数据的变化可以随时由数据源传送至数据仓库，并分别作为插入数据和删除数据保存在数据仓库中。通过这些源数据的变化，可以预先计算出事实表和各个实视图的增量（或变化）。源数据变化的传送和 OLAP 中数据增量的预先计算不影响实视图，因而可在数据仓库服务的同时进行视图更新，以缩短数据仓库因维护而中止其服务的时间。

本章节提出利用视图表达式树中的中间结点来生成辅助视图，并在数据仓库中进行实体化。采用增量维护算法对实视图和辅助视图进行更新维护，实现数据仓库视图增量更新维护。

视图计算可以用一个表达式树来表示，其叶结点表示基关系，非叶结点表示辅助视图、实视图或基关系的二元表达式、以及分组和聚集函数。视图在执行前可以通过查询优化器进行优化，查询优化器将表达式树作为输入，产生一个优化的视图表达式树。

定义 5.5 叶结点的深度是 0，一个结点的深度 d 是子孙结点深度最大值加 1。优化的视图表达式树高度 h 是树中所有结点中的最大深度。

定义 5.6 假设给定表达式树的一个结点 i，其父结点用 $\uparrow i$ 表示，与 i 和 $\uparrow i$ 相关的表达式分别为 op(i)和 op($\uparrow i$)。结点 i 的左右孩子结点分别为 i' 和 i''，其中 i' 是 i'' 的兄弟。IR_i 表示结点 i 的中间结果，结点 i 的辅助视图是 AR_i（只需一个关系时）或 AR_i^1 和 AR_i^2（需要两个关系时）。

定义 5.7 假设结点 i 是树任一结点，结点 i 的中间表达式(IR_i)的精确变化用 δEC_i 来表示（其中插入时 δ 为△，删除时 δ 为▽）；设 AR_i^{old} 和 AR_i^{new} 表示的更新前后的辅助视图，AR_i 的关键字为 K。

定义 5.8 如果满足下列条件，则视图 V 的插入△V 满足精确变化特性：

（1）$V^{new} = V^{old} \cup \triangle V$

（2）$\triangle V \cap V^{old} = \Phi$

定义 5.9 如果满足下列条件，则视图 V 的删除▽V 满足精确变化特性：

（1）$V^{new} = V^{old} - \nabla V$

（2）$\nabla V \cap V^{new} = \Phi$

定义 5.10 视图中聚集函数是用来计算一组元组的某一属性值的聚集值。如果聚集函数在更新数据时能由源数据的更新值和聚集函数旧值推导出聚集函数的新值，称为聚集函数是可增量维护的。

5.2.2 实视图增量更新维护的体系结构

图 5-5 为实视图增量更新和一致性维护的体系结构。在这个体系结构中，保持数据仓库与基础数据源一致。当基础数据源更新时，Monitor 检测变化并将更新信息发送给 Integrator，由 Integrator 从数据源集成必需信息，来更新数据仓库中的视图。

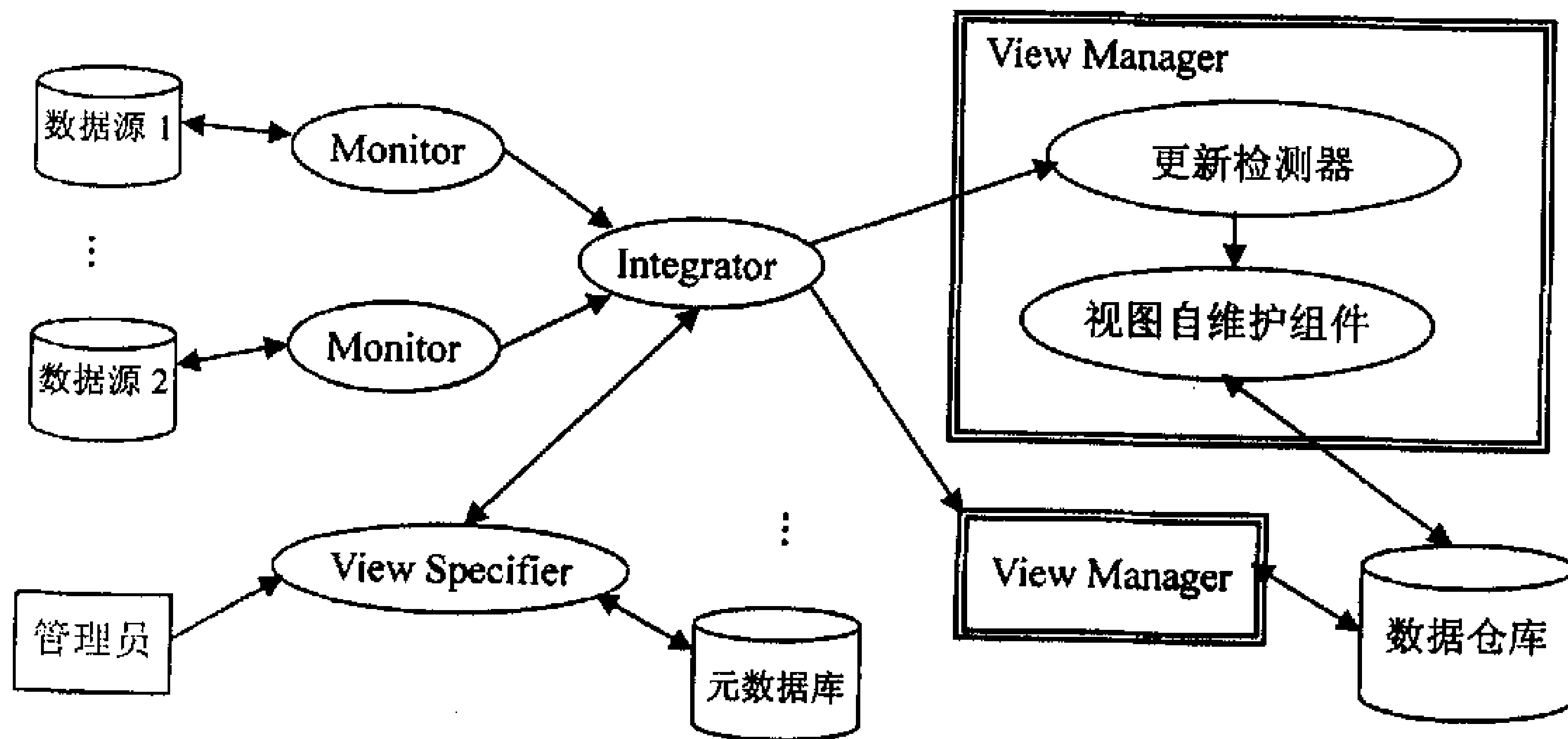


图 5-5 实视图增量更新维护的体系结构

当基础数据源更新时，Integrator 将从 Monitor 收到的更新信息，发送给所涉及的视图管理器（View Manager）。由 View Manager 采用视图增量维护算法，对实视图和辅助视图进行相应地更新，从而使决策者及时从数据仓库实视图得到正确的决策信息。

View Specifier 组件用于管理员对实视图进行操作，检查定义实视图的语法错误，同时向元数据库中写入实视图定义，并将视图定义情况通知到 Integrator。

Integrator 在系统初始化时，对 Monitor 进行登记注册，将登录信息保存起来。在视图初始化时，接收来自 View Specifier 组件对实视图的插入和删除情况，生成一个 View Manager 来创建实视图，将实视图存储到数据仓库中。并将来自 Monitors 的更新信息，发送给相应的 View Manager，进行视图的更新维护。

View Manager 中的更新检测器检测更新是否影响到实视图。如果更新不影响任何视图，这个组件不将这个信息传给其它组件。这时，系统的性能就可以得到很大地改善。如果更新与视图相关，则将更新传给视图增量更新维护组件。视图增量更新维护组件将与实视图更新有关的信息，用辅助视图保存到数据仓库中，通过直接访问存储在数据仓库中的辅助视图，来对实视图进行增量更新维护。实现了数据仓库与基础数据源保持数据一致，同时避免了数据仓库与基础信息之间的网络交换。

5.2.3 视图增量更新维护

视图增量更新维护主要是通过创建相应的辅助视图，并在数据仓库中进行实体化。根据基础数据源的变化对辅助视图进行更新，由辅助视图对实视图进行增量更新。

5.2.3.1 辅助视图的创建与实体化

数据仓库中视图的计算和更新可以采用视图表达式树以由下向上的方式来执行。因而，可以从视图表达式树的中间结果导出辅助视图。为了实现实视图增量更新维护，对导出的辅助视图在数据仓库中进行实体化。辅助视图的创建和实体化主要步骤如下：

（1）当 op(i) = “⋈” 时，如果 op(↑i) = “⋈”，则实体化 IR_i。如果 op(↑i) ≠“⋈”，则实体化 $AR_i = \pi_{K_i}(IR_{i'} \bowtie IR_{i''})$。如果连接结点的孩子是基关系，则实体化这个基关系。

（2）当 op(i) = “—” 时，如果 op(↑i) = “⋈”或者结点 i 的关键字和它的孩子结点的关键字不相等时，则实体化 $AR_i^1 = IR_{i'} - IR_{i''}$ 和 $AR_i^2 = IR_{i''} - IR_{i'}$，否则实体化 $AR_i^1 = \pi_{K_{i'}}(IR_{i'}) - \pi_{K_{i''}}(IR_{i''})$ 和 $AR_i^2 = \pi_{K_{i''}}(IR_{i''}) - \pi_{K_{i'}}(IR_{i'})$。

（3）当 op(i) = “∪”时，如果 op(↑i) ≠ “⋈” 或者结点 i 的关键字和它的孩子结点的关键字相等时，则实体化 $AR_i = \pi_{K_i}(IR_i)$ 的关键字属性和元组计数属性 CNT。否则实体化 IR_i 和计数属性 CNT。

（4）当 op(i) = “*Group By*”时，则实体化 IR_i 的分组聚集结果和元组计数属性 CNT。

5.2.3.2 视图增量维护算法

视图的更新可以通过视图表达式树来执行，利用视图表达式树由下向上的方式来计算视图的变化。也就是说，在视图表达式树先根据叶结点的基关系更新情况，计算出中间结点的变化，然后依次以由下向上的方式计算每个中间结点的增量更新情况，直至某个祖先结点不更新时或根结点（即最终视图）为止。树中各结点可能是连接、并、差和分组四种类型结点，每个结点的变化来自于其孩子结点以及为这些孩子和结点本身实体化的辅助视图的变化。这里，提出一种视图增量维护算法，在不访问视图本身来计算每个中间结点和视图的精确变化。该算法利用视图表达式树中的孩子结点和实体化辅助视图的结果，来计算视图的精确变化，对辅助视图和实视图进行增量更新维护。

视图增量维护算法如算法 5.2 所示。

算法 5.2：视图增量维护算法

{/*输入：视图表达式树，增量（δ）关系，输出：更新后的辅助视图和实视图*/

for each "delta" relation update, say, to R_i

找出表达式树中 R_i 的所有只需更新的祖先结点 N;

for (d = 1;d<= h;d++)

{for (each node i in N having depth d)

{ switch(op(i))

{Case "⋈":

{if（i' 更新) {$Temp1 = \delta AR_{i'}$; $Temp2 = \pi_{A_{i'},A_{i''}}(Temp1 \bowtie AR_{i''})$;

if ($|Temp2| = 0$) { $\delta EC_i = \{\}$; $AR_i^{new} = AR_i^{old}$;}

else{$Temp3 = AR_{i''}$ SJ $Temp2$;

$Temp4 = \pi_{A_{i'},A_{i''}}(Temp3 \bowtie AR_{i'}^{old})$; $\delta EC_i = Temp2 - Temp4$;}

/* $A_{i'}, A_{i''}$ 分别表示结点 i', i''非连接属性*, SJ:半连接 */

if($\delta = \triangle$) $AR_i^{new} = AR_i^{old} \cup \Delta EC_i$;

if($\delta = \nabla$) $AR_i^{new} = AR_i^{old} - \nabla EC_i$;}

if(δ=修改) 分解成删除和插入操作，分别按上述的 δ=▽和 δ=△进行更新;}

if（i'' 更新) 执行更新同 i'一样; }

Case "∪":

{ if（i' 更新)

{if(δ=△) { for(each t.K $\in (\pi_K(\Delta EC_{i'}) - \pi_K(AR_i^{old}))$){ $t.K = \Delta EC_{i'}.K$; $t.A = \Delta EC_{i'}.A$;

$\pi_{K,A}(\Delta EC_i) = \pi_{K,A}(\Delta EC_i) \cup \{(t.K, t.A)\}$;}

/*t 为元组，K、A 分别为关键字和非关键字属性*/

if($t.K \in (\pi_K(\Delta EC_i) \cap \pi_K(AR_i^{old}))$) $AR_i^{new}.CNT = 2$;

if($t.K \in (\pi_K(\Delta EC_{i'}) - \pi_K(AR_i^{old}))$ 向 AR_i 插入元组 t 且 $AR_i^{new}.CNT = 1$;}

if(δ=▽) {for (each $t.K \in (\pi_K(\nabla EC_{i'}) \cap \pi_K(AR_i^{old}))$){ $t.K = \nabla EC_{i'}.K$; $t.A = \nabla EC_{i'}.A$;

$\pi_{K,A}(\nabla EC_i) = \pi_{K,A}(\nabla EC_i) \cup \{(t.K, t.A)\}$;}

if($t.K \in (\pi_K(\nabla EC_i) \cap \pi_K(AR_i^{old}))$ and $AR_i^{old}.CNT = 2$) $AR_i^{new}.CNT = 1$

else 从 AR_i 删除元组 t;}

if(δ=修改) 分解成删除和插入操作，分别按上述的 δ=▽和 δ=△进行更新;}

if（i'' 更新) 执行更新同 i'一样; }

Case "−":

{if（i' 更新)

{if (δ =△) {for (each $t.K \in (\pi_K(\Delta EC_{i'}) - \pi_K(AR_i^{2(old)}))$){ $t.K = \Delta EC_{i'}.K$; $t.A = \Delta EC_{i'}.A$;

$\pi_{K,A}(\Delta EC_i) = \pi_{K,A}(\Delta EC_i) \cup \{(t.K, t.A)\}$;}

$AR_i^{1(new)} = AR_i^{1(old)} \cup \{(t.K \in \pi_K(\Delta EC_i), t.A)\}$;

$AR_i^{2(new)} = AR_i^{2(old)} - \{(t.K \in \pi_K(\Delta EC_{i'}), t.A)\}$;}

if(δ =▽) { for(each $t.K \in (\pi_K(\nabla EC_{i'}) \cap \pi_K(AR_i^{1(old)}))$){ $t.K = \nabla EC_{i'}.K$; $t.A = \nabla EC_{i'}.A$;

$\pi_{K,A}(\nabla EC_i) = \pi_{K,A}(\nabla EC_i) \cup \{(t.K, t.A)\}$;}

$AR_i^{1(new)} = AR_i^{1(old)} - \{(t.K \in \pi_K(\nabla EC_i), t.A)\}$;

$AR_i^{2(new)} = AR_i^{2(old)} \cup \{(t.K \in \pi_K(\nabla EC_i), t.A)\}$; }

if (δ =修改) 分解成删除和插入操作，分别按上述的 δ =▽和 δ =△进行更新; }

if (i'' 更新) 分别将 i'' 的删除和插入当成对 i' 的插入和删除; }

Case "Group By":

{ $\pi_{G,F}(\delta EC_i) = \pi_{G,F}(\delta EC_{i'} \bowtie AR_i^{old})$;

for(each $t.G \in (\pi_G(\delta EC_{i'}) - \pi_G(AR_i^{old}))$){ $t.G = \delta EC_{i'}.G$; $t.F = \delta EC_{i'}.F$;

$\pi_{G,F}(\delta EC_i) = \pi_{G,F}(\delta EC_i) \cup \{(t.G, t.F)\}$;}

for (each $t.G \in (\pi_G(\delta EC_i) \cap \pi_G(AR_i^{old}))$){ $t.G = \delta EC_i.G$; $t.F = \delta EC_i.F$;

$\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) \cup \{(t.G, t.F)\}$;}

for (each $t.G \in (\pi_G(\delta EC_i) - \pi_G(AR_i^{new}))$){ $t.G = \delta EC_i.G$; $t.F = \delta EC_i.F$;

if(δ =△) $\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) \cup \{(t.G, t.F)\}$;

if(δ =▽) $\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) - \{(t.G, t.F)\}$;}}}

/*G，F 为分组属性和聚集函数值*/

if(i 是根结点) 更新实视图; }}

视图增量维护算法主要步骤如下：

1）从视图表达式树中更新的叶结点（基关系）开始，根据表达式树中各结点的不同操作类型分别对更新结点的更新值进行计算，计算出中间结点的变化，对其生成的视图进行增量更新；

2）根据视图表达树中各结点的不同操作类型，分别对其辅助视图和实视图进行增量更新维护；

（1）如果结点 i 是连接操作时（即 op(i) = "⋈"），则根据其更新的孩子结点 i' 或 i'' 的更新值生成的辅助视图，与另外的孩子结点辅助视图进行半连接操作，将半连接操作后的结果对结点 i 的视图进行增量更新；

（2）当结点 i 是并操作时（即 op(i) = "∪"），则根据在孩子结点进行元组插入或元组删除等不同类型的数据更新操作，分别进行相应的增量更新操作：

① 如果其孩子结点是进行元组插入操作时，则将关键字属性值在该孩子结点插入更新的数据集 $\Delta EC_{i'}$ 中而不在结点 i 的辅助视图中的元组，直接插入到结点 i 的视图中，同时根据该插入元组的孩子结点个数来修改其元组计数属性 CNT；

② 如果其孩子结点是进行元组删除操作时，则将结点 i 的辅助视图中与该孩子结点删除更新的数据集 $\nabla EC_{i'}$ 中具有相同关键字属性值且元组计数属性 CNT=2 的元组的计数属性 CNT 修改为 1；将具有相同关键字值且元组计数属性 CNT=1 的元组从结点 i 的视图中删除；

③ 如果其孩子结点是进行元组修改操作时，将修改操作分解成删除操作和插入操作，分别按上述元组的删除和插入操作对结点 i 的视图进行增量更新；

（3）如果结点 i 是差操作时（即 op(i) = "－"），则根据在不同孩子结点上进行的元组更新操作，分别对结点 i 的辅助视图进行增量更新操作。如果是在左孩子结点 i' 插入元组时，则分别将关键字属性值在左孩子结点 i' 插入更新的数据集 $\Delta EC_{i'}$ 中而不在结点 i 的右辅助视图 $AR_i^{2(old)}$ 中的元组，插入到结点 i 的左辅助视图中，并从其右辅助视图中删除该元组。如果是在左孩子结点 i' 删除元组时，则分别将关键字属性值既在左孩子结点 i' 删除更新的数据集 $\nabla EC_{i'}$ 中又在结点 i 的左辅助视图 $AR_i^{1(old)}$ 中的元组，插入到结点 i 的右辅助视图中，并从其左辅助视图中删除该元组。（如果是对 i'' 进行更新，同 i' 一样进行上述的更新操作）；

（4）如果结点 i 是分组操作时（即 op(i) = "*Group By*"），则在其孩子结点的更新数据集 $\delta EC_{i'}$ 上运用分组表达式运算。将分组属性值只在 $\delta EC_{i'}$ 中而不在结点 i 的辅助视图 AR_i 中的元组追加到 δEC_i 中，用 δEC_i 来更新 AR_i，并将分组属性值在 δEC_i 中而不在 AR_i 中的元组插入到 AR_i 中或从 AR_i 中删除。

3）根据视图表达式树以由下向上的方式，依次对所有受到更新影响的祖先结点按步骤 2）对视图进行增量更新维护，直至某个祖先结点不再更新时或根结点（即最终视图）为止。

本算法是对视图表达式树中各种不同类型结点进行更新维护的增量计算，满足在不访问基础数据就可以计算出其精确变化，实现了数据仓库视图增量更新维护和一致性维护。

由于数据仓库中聚集函数的更新是视图更新维护中最重要且最复杂的，视图中聚集函数的更新见下一节 5.2.3.3。

5.2.3.3 聚集函数的增量更新算法

假设结点 i 的操作中含有聚集函数（如 AVG、COUNT、MAX、MIN 或 SUM），如果其孩子结点 i' 数据发生变化时，将在结点 i' 这个分组结点上执行一组查询来计算聚集函数变化，将这些变化传给结点 i。由于 i' 是 i 的孩子，因此 i' 的更新一定被传播到 i 上。根据结点 i 的所含聚集函数类型不同，其更新计算分为以下两种情况（如果 δ 是△（插入）则用"＋"，否则用"－"）：

1） SUM、AVG 或 COUNT 聚集函数

① 在 $\delta EC_{i'}$ 上运用分组表达式运算，将计数结果赋给 AGG_CNT，并保存到 $\delta EC_{i'}$ 中。

② 对 AR_i 和 $\delta EC_{i'}$ 中具有相同分组属性值的元组计算其聚集函数值，将计算结果保存到 δEC_i 中。并将分组属性值只在 $\delta EC_{i'}$ 中而不在 AR_i 中的元组追加到 δEC_i 中。

③ 用 δEC_i 更新 AR_i，将 AR_i 中的元组聚集函数值用与其具有相同的分组属性值的 δEC_i 相应的聚集函数值来替换，并将分组属性值在 δEC_i 中而不在 AR_i 中的元组插入到 AR_i 中（δ 为△

时）或从 AR_i 中删除（δ 为▽时）。

2) MIN 或 MAX 聚集函数

（1） 当 i' 的更新是△（插入元组）

① 将在 $\triangle EC_{i'}$ 与 AR_i 中具有相同分组属性值，且前者的聚集函数值大于（或小于）后者相应聚集函数值的元组保存到 $\triangle EC_i$ 中。并将分组属性值在 $\triangle EC_{i'}$ 中而不在 AR_i 中的元组追加到 $\triangle EC_i$ 中。

② 用 $\triangle EC_i$ 更新 AR_i，将 AR_i 中的元组聚集函数值用与其具有相同的分组属性值的 $\triangle EC_i$ 的聚集函数值来替换，并将分组属性值在 $\triangle EC_i$ 中而不在 AR_i 中的元组插入到 AR_i 中。

（2） 当 i' 的更新是▽（删除元组）

① 将 $\nabla EC_{i'}$ 中分组属性和聚集函数值与 AR_i 相同的元组保存到 ∇EC_i 中。

② 从 AR_i 中删除元组 ∇EC_i。将 $AR_{i'}$ 与 ∇EC_i 中分组属性值相同的元组追加到 AR_i 中。

聚集函数的增量更新算法如算法 5.3 所示。

算法 5. 3：聚集函数的增量更新算法

{for each Aggregate Function node{

switch(Type of Aggregate Function)

{case "SUM" or "AVG" or "COUNT":

{ $\delta EC_{i'}.Agg_CNT=COUNT(\pi_G(\delta EC_{i'}))$;

if (Type of Aggregate Function is SUM)

for each $t.G \in (\pi_G(\delta EC_{i'}) \cap \pi_G(AR_i^{old}))$ /*t 为元组*/

$\{t.F = AR_i^{old}.F + (-)\delta EC_{i'}.F ; t.G = \delta EC_{i'}.G ;$

$\pi_{G,F}(\delta EC_i) = \pi_{G,F}(\delta EC_i) \cup \{(t.G, t.F)\} ;\}$

if (Type of Aggregate Function is AVG)

for each $t.G \in (\pi_G(\delta EC_{i'}) \cap \pi_G(AR_i^{old}))$

$\{t.F = \dfrac{AR_i^{old}.F + (-)\delta EC_{i'}.F}{AR_i^{old}.Agg_CNT + (-)\delta EC_{i'}.Agg_CNT} ; t.G = \delta EC_{i'}.G ;$

$\pi_{G,F}(\delta EC_i) = \pi_{G,F}(\delta EC_i) \cup \{(t.G, t.F)\} ;\}$

if (Type of Aggregate Function is COUNT)

for each $t.G \in (\pi_G(\delta EC_{i'}) \cap \pi_G(AR_i^{old}))$

$\{t.F = AR_i^{old}.Agg_CNT + (-)\delta EC_{i'}.Agg_CNT ; t.G = \delta EC_{i'}.G ;$

$\pi_{G,F}(\delta EC_i) = \pi_{G,F}(\delta EC_i) \cup \{(t.G, t.F)\} ;\}$

for each $t.G \in (\pi_G(\delta EC_{i'}) - \pi_G(AR_i^{old}))\{t.G = \delta EC_{i'}.G ; t.F = \delta EC_{i'}.F ;$

$\pi_{G,F}(\delta EC_i) = \pi_{G,F}(\delta EC_i) \cup \{(t.G, t.F)\} ;\}$

for each $t.G \in (\pi_G(\delta EC_i) \cap \pi_G(AR_i^{old}))\{t.G = \delta EC_i.G ; t.F = \delta EC_i.F ;$

$\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) \cup \{(t.G, t.F)\};\}$

for each $t.G \in (\pi_G(\delta EC_i) - \pi_G(AR_i^{new}))\ \{t.G = \delta EC_i.G;\ t.F = \delta EC_i.F;$

if($\delta = \triangle$) $\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) \cup \{(t.G, t.F)\};$

if($\delta = \nabla$) $\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) - \{(t.G, t.F)\};\}\}$

case "MAX" or "MIN" :

{if($\delta = \triangle$) $\{\pi_{G,F}(\Delta EC_i) = \pi_{G,F}(\sigma_{EC_i.F>(<)AR_i^{old}.F}(\Delta EC_i \bowtie AR_i^{old});$

for each $t.G \in (\pi_G(\Delta EC_i) - \pi_G(AR_i^{old}))\ \{t.G = \Delta EC_i.G;\ t.F = \Delta EC_i.F;$

$\pi_{G,F}(\Delta EC_i) = \pi_{G,F}(\Delta EC_i) \cup \{(t.G, t.F)\};\}$

for each $t.G \in (\pi_G(\Delta EC_i) \cap \pi_G(AR_i^{old}))\ \{t.G = \Delta EC_i.G;\ t.F = \Delta EC_i.F;$

$\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) \cup \{(t.G, t.F)\};\}$

for each $t.G \in (\pi_G(\Delta EC_i) - \pi_G(AR_i^{new}))\ \{t.G = \Delta EC_i.G;\ t.F = \Delta EC_i.F;$

$\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) \cup \{(t.G, t.F)\};\}\}$

if($\delta = \nabla$) $\{\pi_{G,F}(\nabla EC_i) = \pi_{G,F}(\sigma_{EC_i.F=AR_i^{old}.F}(\nabla EC_i \bowtie AR_i^{old});$

$\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) - \pi_{G,F}(\nabla EC_i);$

$Temp[i] = \pi_{G,F}(\nabla EC_i \bowtie AR_i^{new});$

$\pi_{G,F}(AR_i^{new}) = \pi_{G,F}(AR_i^{new}) \cup Temp[i];\}\quad \}\}\}$

5.3 本章小结

本章提出根据OLAP查询的选择谓词构造的最小项谓词和访问频率，选择数据仓库视图进行水平分割并实体化。通过查询访问组件预先确定视图及其裂片来快速响应OLAP查询，使大多数查询在分割的实视图上访问，这与在原视图上访问相比仅需扫描更少的元组。从而减少查询响应时间和维护费用，改善了系统的性能。

通过视图表达式树的中间结果来创建辅助视图，并在数据仓库中进行实体化。当基础数据源发生数据更新时，只需通过访问实体化的辅助视图，就能实现数据仓库视图一致性维护；通过基于视图表达式树的视图增量维护算法，实现了在不访问视图本身的情况下就可以导出所有实体化辅助视图和实视图的精确变化，实现数据仓库辅助视图和实视图快速的增量更新维护。这样既保证了视图更新的正确性，也大大地缩短了数据仓库更新维护时间，提高了数据刷新频率和质量。

第六章　结论

6.1　本文的总结

OLAP 技术在基于数据仓库的 DSS 中发挥了关键性的作用。OLAP 常常用来管理决策所需要的总结数据，满足用户的即席（ad hoc）查询，及时向用户提供分析数据，以辅助决策，这就对 OLAP 查询响应速度提出了很高的要求。对数据仓库中的数据进行预聚集处理是提高 OLAP 查询效率的有效途径。作者在参与完成 SEUDW 项目的背景下，着重对提高 OLAP 查询分析效率的若干关键技术进行了系统深入的研究，并取得了以下的主要研究成果：

（1）分析了维属性的层次性以及维层次树中成员具有相同的前缀路径等特点，提出了基于维层次编码的一种新型预分组聚集算法 DHEPGA。DHEPGA 算法利用维层次树创建的维层次编码来代替维表中原关键字，实现维关键字的压缩，大大降低了存储空间。通过维层次编码前缀匹配操作，快速检索出与查询关键字相匹配的维层次编码，求得维层次属性的查询范围，大大减少 I/O 开销。充分利用维层次前缀对事实表中的数据进行快速预分组和层次聚集操作，可以提高事实表中的记录检索速度，大大减少和简化了事实表与维表之间的多表连接，提高了 OLAP 查询效率。算法分析和实验结果表明，DHEPGA 算法性能是非常有效的。同时还可以通过维层次前缀来提高 roll up 和 drill down 等操作效率。

（2）提出了利用 Cube 中的维层次聚集技术来创建高效的维层次聚集 Cube（dimension hierarchy aggregate cube，简称 DHAC）。在用户提交 MOLAP 查询 Q 时，根据查询 Q 的查询空间 MBB_q 与 DHAC 中各结点的 MBB_c 进行比较，来计算 Q 的查询结果，从而提高了 MOLAP 查询效率。在 DHAC 中进行数据插入和删除等数据更新时，利用维层次聚集树中的维层次前缀由下而上对受到更新结点影响的所有祖先结点进行增量更新；在插入新维数据时，不需要重新构建聚集 Cube 就可以对维层次聚集 Cube 进行增量更新，从而提高了 Cube 的更新效率。对 DHAC 与 SDDC、HDC 等聚集 Cube 进行了性能分析和比较，结果表明所提出的 DHAC 性能最佳。

（3）提出了一种数据仓库实视图选择与分割算法。根据 OLAP 查询中的选择谓词构造其最小项谓词，选择数据仓库视图进行有效的水平分割和实体化，使 OLAP 查询通过访问较少的分割裂片以及较少的元组就可以完成，从而加快查询响应时间，削减维护费用，提高了 OLAP 查询效率。

（4）提出了一种数据仓库实视图增量更新维护策略。利用视图计算的中间结果来创建辅助视图，在数据仓库中进行实体化，采用视图表达式树和有效的增量维护算法来计算实视图和辅助视图的精确变化，实现数据仓库视图增量更新维护。解决了数据仓库中存储聚集数据的实视图快速增量更新问题，从而有效地缩短了数据仓库更新维护时间，有效地改善了 OLAP 查询的数据质量和效率。

6.2 进一步的研究工作

提高 OLAP 查询效率和数据质量，仍是当前数据仓库的热点问题。进一步的研究工作有以下几个方面：

（1）在维层次编码与数据仓库系统中现有索引技术的有效结合、维层次编码的有效存储和增量更新、OLAP 查询优化等方面将展开更深入的研究。

（2）采用网格计算等技术实现对聚集 Cube 计算、实视图增量更新维护操作的并行计算，提高 OLAP 的查询效率和数据更新速度。

（3）在本课题组开发的数据仓库系统原型 SEUDW 的基础上，将本文的研究成果实际运用于开发一个高性能多维数据仓库系统。

致谢

值此论文完成之际，首先向我尊敬的导师董逸生教授表示最衷心的感谢！感谢他三年来对我的悉心指导和教导！恩师董教授渊博的学识、严谨的治学态度和他勤奋踏实的为人做事的作风，是我一生学习的榜样和敬仰的长者，也必将使我终身受益。

同时感谢教研室孙志挥教授、王茜教授、徐宏炳教授、徐立臻教授、金远平教授、刘亚军副教授、何洁月副教授、陈钢副教授、姜洁副教授、王伟讲师等诸多老师，他们都曾在不同方面给过我很大的帮助和鼓励。还有教研室和系里其他所有给予我指导或帮助的诸位老师，在此一并表示敬意。

还感谢宋爱波、吴文明、业宁、梁作鹏、杨科华、韩京宇、许卓明、张朝晖、罗广博、钱江波、庄晓清、毕然、王海峰、楼笑、董树明、薛惠忠等师兄弟所给予的帮助和合作。

最后谨以此文献给我最亲爱的妻子陈素娟女士和最可爱的女儿胡恒越！在此也特别感谢我的父母、岳父、岳母给我们的支持和帮助！

参考文献

[1] 王能斌编著. 数据库系统原理[M]. 北京:电子工业出版社, 2000.

[2] 王珊等. 数据仓库技术与联机分析处理[M]. 北京:科学出版社,1998.

[3] 林杰斌,刘明德,陈湘等. 数据挖掘与 OLAP 理论与实务[M]. 北京:清华大学出版社,2003.

[4] Chaudhuri S, Dayal U. An overview of data warehousing and OLAP technology[J]. ACM SIGMOD Record, 1997,26(1):65-74.

[5] Chaudhuri U, Dayal U. Data warehousing and OLAP for decision support[J]. ACM SIGMOD Record, 1997,26(2): 507-508.

[6] Ramakrishnan R. Database Management Systems[M], McGraw-Hill. New York, 1998.

[7] Codd E F. Codd S B, Salley C T. Providing OLAP(On-Line Analytical Processing) to user-analysts: an IT mandate. Technical report. San Jose : Codd & Date,Inc, 1993.

[8] Neil P O, Quass D. Improved query performance with variant indexes[C]. In: Peckham J, ed. Proc. of the ACM SIGMOD International Conference on Management of Data. New York :ACM Press, 1997. 38-49.

[9] Chan C Y, Ioannidis Y E. Bitmap index design and evaluation[C]. In: Haas L M,Tiwary A, eds. Proc. of the ACM SIGMOD International Conference on Management of Data. New York : ACM Press, 1998. 355-366.

[10] Wu M C. Query optimization for selections using bitmaps[C]. In: Delis A, Faloutsos C, Ghandeharizadeh S, eds. Proc. of the ACM SIGMOD International Conference on Management of Data. New York : ACM Press, 1999. 227-238.

[11] Wu K, Otoo E J, Shoshani A. A performance comparison of bitmap indexes[C]. In: Paques H, Liu L, Grossman D, eds. Proc. of the 10th International Conference on Information and Knowledge Management(CIKM). New York :ACM Press, 2001. 559-561.

[12] Bayer R. The universal B-tree for multidimensional indexing: General concepts[C]. In: Masuda T, Masunaga Y, Tsukamoto M,eds. WWCA'97. Heidelberg:Springer,1997. 198-209

[13] Gupta H, Harinarayan V, Rajaranman A,et al. Index selection of OLAP. In: Gray A, Larson P, eds. Proc. of the 13th International Conference on Data Engineering (ICDE). Los Alamitos:IEEE Computer Society Press, 1997. 208-219.

[14] Gupta A, Mumick I S. Maintenance of Materialized views: Problems, techniques, and applications[J]. Data Engineering Bulletin, 1995, 18(2): 3-18

[15] Roussopoulos N. Materialized views and data warehouse[J]. SIGMOD Record ,1998,27(1):21-26

[16] Mistry H, Roy P, Sudarshan S. Materialized view selection and maintenance using multi-query optimization[C]. In: Aref W G, ed. Proc. of the ACM SIGMOD 2001. New York : ACM Press. 2001. 307-318.

[17] Sameet A, Rakesh A, Deshpande P, et al. On the computation of multidimensional aggregates[C]. In: Vijayaraman T M, Buchmann A P, Mohan C,eds. Proc. of the 22nd International Conf on VLDB, San Fransisco:Morgan Kaufmann ,1996. 506-521.

[18] Markl V, Ramsak F , Bayern R. Improving OLAP performance by multidimensional hierarchical Clustering[C]. In: Adiba M, Collet C, Desai B C, eds. Proc. of the International Conference on Database Engineering and Applications(IDEAS 1999). Los Alamitos:IEEE Computer Society Press, 1999. 165-177.

[19] Kotidis Y, Roussopoulos N. An alternative storage organization for ROLAP aggregate views based on cubetrees. In: Haas L M, Tiwary A,eds.Proc. ACM SIGMOD International Conference on Management of Data. New York :ACM Press,1998. 249-258.

[20] Galindo-Legqria C A. Joshi M. Orthogonal optimization of subqueries and aggregation[J]. ACM SIGMOD Record , 2001,30(2): 571- 581.

[21] Zou C, Salzberg B, Ladin R. Back to the future: dynamic hierarchical clustering[C]. In:. Proc. of the 4th International Conference on Data Engineering(ICDE). Los Alamitos:IEEE Computer Society Press,1998:578-587.

[22] Haas L M, Carey M J, Livny M, et al. Seeking the truth about ad hoc join costs[J]. VLDB Journal 1997,6(3):241-256.

[23] Harris E P, Ramamohanarao K. Join algorithm costs revisited[J].VLDB Journal ,1996,5(1):64-84.

[24] Yan W P, Larson P A. Eager aggregation and lazy aggregation[C]. In: Dayal U, Gray P M D, Nishio S,eds . Proc. of the 21th International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann, 1995. 345-357.

[25] Goerkotte G. Small materialized aggregates: a light weight index structure for data warehousing[C]. In: Atkinson M P, Orlowska M E, Valduriez P, eds. Proc. of the 25th International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann, 1999. 476-487.

[26] Markl V, Zirkel M, Bayer R. Processing operations with restrictions in relational database management systems without external sorting:the Tetris algorithm[C]. In: Kitsuregawa M, eds. Proc. of the 15th International Conference on Data Engineering (ICDE). Los Alamitos:IEEE Computer Society Press, 1999. 562-571.

[27] Golfarelli M, Rizzi S. A methodological framework for data warehouse design[C]. In: Song I Y, Teorey T J, eds. Proc. of the 1st International Workshop on Data Warehousing and OLAP(DOLAP). New York :ACM Press,1998. 3-9.

[28] Lehner W. Modeling large scale OLAP scenarios[C]. In: Schek H J, Saltor F, Ramos I, eds. Proc. of the 6th International Conference on Extending Database Technology(EDBT). Heidelberg:Springer, 1998. 153-167

[29] Tryfona N, Busborg F, Chistiansen J. starER: a conceptual model for data warehouse design[C]. In: Song I Y, Teorey T J, eds. Proc. of the 2nd International Workshop on Data Warehousing and OLAP(DOLAP). New York :ACM Press, 1999. 3- 8.

[30] Vassiliadis P, Sellis T K. A survey of logical models for OLAP databases[J]. ACM SIGMOD Record,1999, 28(4): 64-69.

[31] 蒋旭东,冯建华,周立柱. 联机分析查询处理中的一种聚集算法[J]. 软件学报, 2002,13(1) :65- 70

[32] 冯建华,蒋旭东,孟宪虎. 基于分组序号的聚集算法[J].软件学报,2003,14(4):222- 228.

[33] Theodoratos D, Tsois A. Heuristic optimization of OLAP queries in multidimensionally hierarchically clustered databases[C]. In: Hammer J, ed. Proc. of the 4th ACM International Workshop on Data Warehousing and OLAP , New York : ACM Press,2001. 48 – 55.

[34] Karayannidis N, Tsois A, Sellis T, et al. Processing Star Queries on Hierarchically-Clustered Fact Tables[C]. In: Bressan S, Chaudhri A B, Lee M L, eds. Proc. of the 28th International Conf on VLDB. San Fransisco:Morgan Kaufmann, 2002.730~741.

[35] Tsois A, Karayannidis N, Sellis T,Theodoratos D. Cost-based optimization of aggregation star queries on hierarchically-clustered fact tables[C]. In: Lakshmanan L V S, ed. Proc. of the 4th International Workshop on Design and Management of Data Warehouses(DMDW). Germany:Technical University of Aachen (RWTH),2002. 62-71.

[36] Gray J, Bosworth A, Layman A, Pirahesh H. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals[C]. In: Su S Y W ,ed. Proc. of the 12th International Conference on Data Engineering. Los Alamitos:IEEE Computer Society Press, 1996. 131-139

[37] Zhao Y, Deshpande P M, and Naughton J F. An array-based algorithm for simultaneous multidimensional aggregates[C]. In: Peckham J, ed. Proc. of the ACM SIGMOD Conference 1997. New York :ACM Press,1997. 159-170.

[38] Mumick I S, Quass D, Mumick B S. Maintenance of data Cubes and summary tables in a warehouse[J]. ACM SIGMOD Record,1997,26(2): 100-111.

[39] Gaede V, Gunther O. Multidimensional access methods[J]. ACM computing surveys,1997, 30(2):170-231.

[40] Deshpande P, Ramasamy K, Shukla A, et al. Caching multidimensional queries using chunks[C].In: Haas L M, Tiwary A,eds. Proc. ACM SIGMOD International Conference on Management of Data. New York :ACM Press,1998.259-270.

[41] Cabibbo L, Torlone R. A logical approach to multidimensional databases[C]. In: Schek H J, Saltor F, Ramos I, eds. Proc. of the 6th International Conference on Extending Database Technology(EDBT). Heidelberg:Springer,1998. 183-197.

[42] Barbara D, Sullivan M. Quasi-cubes: Exploiting approximation in multidimensional database[J]. ACM SIGMOD Record, 1997,26:12-17.

[43] Ross K , Srivastava D. Fast Computation of sparse datacubes. In: Jarke M, Carey M J, Dittrich K R, eds. Proc. of the 23rd International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann, 1997. 116-125.

[44] Sarawagi S, Agrawal R, Megiddo N. Discovery- Driven Exploration of OLAP Data Cubes. In: Schek H J, Saltor F, Ramos I, eds. Proc. of the 6th International Conference on Extending Database Technology(EDBT). Heidelberg:Springer, 1998. 168-182.

[45] Vitter J S, Wang M, Iyer B R. Data Cube approximation and histograms via wavelets. In: Gardarin G, French J C, Pissinou N,eds. Proc. of the 7th International Conference on Information and Knowledge Management(CIKM). New York :ACM Press,1998. 96-104.

[46] Trujillo J. The GOLD Model: An OO Multidimensional Data Model for Multidimensional Databases[C]. In: Guerraoui R,ed. Proc. of the 13th European Conference on Object-Oriented Programming(ECOOP). Heidelberg:Springer, 1999. 24-30.

[47] Vassiliadis P. Modeling multidimensional databases, cube and cube operations[C]. In: Rafanelli M, Jarke M,eds. Proc. of the 10th International Conference on Scientific Statistical Database Management(SSDBM). Los Alamitos:IEEE Computer Society Press, 1998. 53-62.

[48] Beyer K, Ramakrishnan R. Bottom-up computation of sparse and iceberg cubes.In: Delis A, Faloutsos C, Ghandeharizadeh S,eds. Proc. of the ACM SIDMOD International Conference on Management of Data. New York :ACM Press,1999. 359-370.

[49] Han J, Pei J, Dong G, et al. Efficient computation of iceberg cubes with complex measures. In: Aref W G, ed. Proc. of the ACM SIGMOD Conference on Management of Data. New York :ACM Press,2001. 1-12.

[50] Sathe G, Sarawagi S. Intelligent rollups in multidimensional OLAP data.In: Apers P M G, Atzeni P, Ceri S, eds. Proc. of the 27th International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann, 2001. 531-540.

[51] Chou P, Zhang X. Efficiently computing the top N averages in iceberg cubes. In: Oudshoorn M, ed. Proc. of the twenty-sixth Australasian Computer Science Conference on Conference in Research

and Practice in Information Technology(ACSC). Australia: Australian Computer Society, 2003. 101 – 109.

[52] Harinarayan V, Rajaraman A, Ullman J D. Implementing data cubes efficiently. In: Jagadish H V, Mumick I S, eds. Proc. of the ACM SIGMOD International Conference on Management of Data. New York :ACM Press,1996. 205-216.

[53] Ho C, Agrawal R, Megiddo N, Srikant R. Range queries in OLAP data Cubes[J]. ACM SIGMOD Record,1997,26(2): 73-88.

[54] Geffner S, Agrawal D, Abbadi A E1, et al. Relative prefix sums: An efficient approach for querying dynamic OLAP data Cubes[C].In: Kitsuregawa M, eds. Proc. of the 15th International Conference on Data Engineering (ICDE). Los Alamitos:IEEE Computer Society Press, 1999.328-335.

[55] Geffner S, Agrawal D, Abbadi A E1. The dynamic data Cubes[C]. In: Zaniolo C, Lockemann P C, Scholl M H, eds. Proc. of 7th International Conference on Extending Database Technology(EDBT). Heidelberg:Springer, 2000. 237-253.

[56] Liang W, Wang H, Orlowska M E. Range queries in dynamic OLAP data Cubes[J]. Data & Knowledge Engineering , 2000,34(1):21-38.

[57] Riedewald M, Agrawal D, Abbadi A El. Flexible Data Cubes for Online Aggregation[C]. In: Bussche J V, Vianu V, eds. Proc. of the 8th International Conference on Database Theory (ICDT'01). Heidelberg:Springer, 2001. 159-173.

[58] Karayannidis N, Sellis T. SISYPHUS: A chunk-based storage manager for OLAP cubes. In: Theodoratos D, Hammer J, Jeusfeld M A,eds. Proc. of the 3rd International Workshop on Design and Management of Data Warehouses(DMDW). Germany:Technical University of Aachen (RWTH), 2001.10

[59] Theodoratos D, Sellis T. Answering queries on cubes using other cubes. In: Günther O, Lenz H, eds. Proc. of the 12th International Conference on Scientific and Statistical Database Management (SSDBM). Los Alamitos:IEEE Computer Society Press, 2000.109-122.

[60] Li J Z, Srivastava J. Efficient aggregation algorithms for compressed data warehouses[J]. IEEE Trans on Knowledge and Data Engineering, 2002,14(3):515-529.

[61] 高宏,李建中,李金宝. 数据仓库系统中层次式 Cube 存储结构[J]. 软件学报, 2003,14(7) : 1258-1266.

[62] 冯建华,蒋旭东,周立柱. 用于数据仓储的一种改进的多维存储结构[J]. 软件学报, 2002,13(8):1423- 1429.

[63] Feng Y, Wang S. Compressed data Cube for approximate OLAP query processing[J]. Journal of Computer Science and Technology, 2002,17(5):625-635.

[64] Wang W, Lu H J, Feng J L, et al. Condensed Cube: An Effective Approach to Reducing Data cube Size[C]. .In: Jose S, Agrawal C R, Dittrich K, eds. Proc. of the 18th International Conference on Data Engineering (ICDE). Los Alamitos:IEEE Computer Society Press, 2002. 155-165.

[65] Sismanis Y, Deligiannakis A, Roussopoulos N, et al. Dwarf: Shrinking the PetaCube[C]. In: Voisard A, Chen S C, eds. Proc. of the ACM SIGMOD International Conference on Management of Data. New York :ACM Press, 2002. 464-475.

[66] Shanmugasundaram J, Fayyad U M, Bradley P S. Compressed Data Cubes for OLAP Aggregate Query Approximation on Continuous Dimension. In: Chaudhuri S, Madigan D, eds. Proc. of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). New York :ACM Press,1999. 223-232.

[67] Barbara D, Wu X. Using loglinear models to compress datacube.In: Lu H J, Zhou A Y,eds.Proc. of the 1st International Conference on Web-Age Information Management(WAIM). Heidelberg:Springer,2000. 311-322.

[68] Li J Z, Riotem D, Srivastava J. Aggregation algorithms for very large compressed data warehouse[C]. In: Atkinson M P, Orlowska M E, Valduriez P, eds.Proc. of the 25th International Conference on Very Large Data Bases. San Fransisco: Morgan Kaufmann,1999. 651-662.

[69] Lakeshmanan L V S, Pei J, Han J. Quotient Cubes: How to Summarize the Semantics of a Data Cube. In: Bressan S, Chaudhri A B, Lee M L, eds. Proc. of the 28th International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann,2002. 778-789.

[70] Lakeshmanan L V S, Pei J, Zhao Y. QC-Trees: An Efficient Summary Structure for Semantic OLAP. In: Halevy A Y, Ives Z G, Doan A,eds. Proc. of the ACM SIGMOD Conference on Management of Data. New York :ACM Press,2003. 64-75.

[71] Hurtado C A, Mendelzon A O, Vaisman A A. Maintaining data cubes under dimension updates. In: Kitsuregawa M, eds. Proc. of the 15th International Conference on Data Engineering (ICDE). Los Alamitos:IEEE Computer Society Press, 1999 .346-355.

[72] Hurtado C A, Mendelzon A O, Vaisman A A. Updating OLAP dimensions. In . In: Song I Y, Teorey T J, eds. Proc. of the 2nd International Workshop on Data Warehousing and OLAP(DOLAP). New York :ACM Press,1999.60-66.

[73] Chun S J, Chung C W, , J. H. Lee, et al. Dynamic update cube for range-sum queries[C]. In: Apers P M G, Atzeni P, Ceri S, eds. Proc. of the 27th International Conference on the Very Large Data Base. San Fransisco: Morgan Kaufmann, 2001. 521-530.

[74] Widom J. Research problems in data warehousing. In: Qaigi Z, Ziengliang L,eds. Proc. of the 4th International Conference on Information and Knowledge Management(CIKM). New York :ACM Press, 1995.25-30.

[75] Cohen S, Nutt W, Serebrenik A. Rewriting aggregate queries using views. In: Papadimitriou C,ed. Proc. of the 18th Symposium on Principles of Database Systems. New York :ACM Press,1999. 155-166.

[76] Fang M, Shivakumar N, Molina G H, et al. Computing iceberg queries efficiently. In: Gupta A, Shmueli O, Widom J, eds. Proc. of the 24th International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann,1998. 299-310.

[77] Balmin A, Papadimitriou T, Papakonstantinou.Y. Hypothetical queries in an OLAP environment. In: Abbadi A El, Brodie M L, Chakravarthy S, eds. Proc. of the 26th International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann, 2000. 220-231.

[78] Mendelzon A O , Vaisman A A. Temporal queries in OLAP. In: Abbadi A El, Brodie M L, Chakravarthy S, eds. Proc. of the 26th International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann, 2000. 242-253.

[79] Datta A, Ramamritham K, Thomas H. A novel solution for efficient storage and indexing in data warehouses[C]. In: Atkinson M P, Orlowska M E, Valduriez P,eds. Proc. of the 25th International Conference on Very Large Datadases. San Fransisco:Morgan Kaufmann ,1999. 730-733.

[80] Baralis E, Paraboschi S, Teniente E. Materialized view selection in a multidimensional database[C]. In: Jarke M, Carey M J, Dittrich K R, eds. Proc. of the 23rd International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann, 1997. 156-165.

[81] Gupta H. Selection of views to materialize in a data warehouse.In: Afrati F N, Kolaitis P G, eds. Proc. of the 6th International Conference on Database Theory(ICDT). Heidelberg:Springer, 1997. 98-112.

[82] Shukla A, Deshpande P M, Naughton J F. Materialized view selection for multidimensional datasets. In: Gupta A, Shmueli O, Widom J, eds. Proc. of the 24th International Conference on Very Large Data Bases(VLDB). San Fransisco: Morgan Kaufmann,1998. 488-499.

[83] Chaudhuri S, Narasayya V. Index merging[C]. In: Kitsuregawa M, eds. Proc. of the 15th International Conference on Data Engineering(ICDE) . Los Alamitos:IEEE Computer Society Press, 1999. 296-303.

[84] Noamam A Y, Barker K. A horizontal fragmentation algorithm for the fact relation in a distributed data warehouse[C]. In: Gauch S, ed. Proc. of the 8th International Conference on Information and Knowledge Management(CIKM). New York :ACM Press ,1999. 154-161.

[85] Ozsu M T, Valduriez P. Principles of Distributed Database systems[M]. Englewood: Prentice-Hall ,1999.

[86] Bellatreche L, Karlapalem K, Mohania M, et al. What can partitioning do for your data warehouses and data marts?[C] In: Desai B C, Kiyoki Y, Toyoma M, eds. Proc. of the International Database Engineering and Applications Symposium (IDEAS). Los Alamitos:IEEE Computer Society Press, 2000. 437-445.

[87] Bellatreche L, Karlapalem K, Mohania M. OLAP query processing for partitioned data warehouses[C]. In: Kambayashi Y, Takakura H, eds. Proc. of the International Symposium on Database Applications in Non-Traditional Environments(DANTE). Los Alamitos:IEEE Computer Society Press, 1999. 28-30.

[88] Zhuge Y, Gracia-Molina H, Wiener J. The Strobe algorithms for multi-source warehouse consistency. In: Santoro N, Widmayer P, eds. Proc. of the 4th International Conference on Parallel and Distributed Information Systems (PDIS). Los Alamitos:IEEE Computer Society Press, 1996.146-157.

[89] Zhuge Y, Gracia-Molina H, Wiener J. Consistency algorithms for multi-source warehouse view maintenance[J]. Journal of Distributed and Parallel Databases, 1998,6(1):7-40.

[90] Agrawal D, Abbadi A El, Singh A,et al. Efficient view maintenance at data warehouses[C]. In: Huhns M N, Singh M P, eds. Proc. of the ACM SIGMOD International Conference on Management of Data. New York :ACM Press, 1997.417-427.

[91] Hull R, Zhou G. A framework for supporting data integration using the materialized and virtual approaches[C]. In: Jagadish H V, Mumick I S, eds. Proc. of the ACM SIGMOD Conference On Management of Data. New York :ACM Press,1996. 481-492.

[92] Dallan Q, Ashish G, Indelpal S M, et al. Making views self-maintainable for data warehousing[C]. In: Widmayer P, eds. Proc. of International Conference on Parallel and Database Information Systems. Los Alamitos:IEEE Computer Society Press, 1996. 158-169.

[93] Ross K A, Srivastava D, Sudarshan S. Materialized view maintenance and integrity constraint checking:trading space for time[C]. In: Jagadish H V, Mumick I S, eds. Proc. of the ACM SIGMOD International Conference on Management of Data. New York :ACM Press,1996. 447 – 458.

[94] Bailay J, Dong G, Mohania M, et al. Distributed view maintenance by incremental semijion and tagging[J]. Distributed and Parallel Databases, 1998, 6(3):287-309.

[95] Jaedicke M, Mitschang B. On parallel processing of aggregate and scalar functions in object-relational DBMS[C]. In: Haas L M, Tiwary A,eds. Proc. of the ACM SIDMOD International Conference on Management of Data. New York :ACM Press,1998.379-389.

[96] Colby L, Kawaguchi A, Lieuwen D, et al. Supporting multiple view maintenance policies[C]. In: Peckham J, ed. Proc. of the ACM SIGMOD Conference on Management of Data. New York :ACM Press,1997. 405-416.

[97] Yang J, Widom J. Temporal view self-maintenance. In: Zaniolo C, Lockemann P C, Scholl M H, eds. Proc. of 7th International Conference on Extending Database Technology(EDBT). Heidelberg:Springer,2000. 395-412.

[98] Huyu N. Multiple view self-maintenance in data warehousing environments[C]. In: Jarke M, Carey M J, Dittrich K R, eds. Proceeding of the 23rd International Conference on Very Large Database. San Fransisco: Morgan Kaufmann, 1997. 26-35.

[99] Liang W,Wang H,Orlowska M E, et al. Making multiple views self-maintainable in a data warehouse[J]. Data & Knowledge Engineering, 1999, 30(2):121-134.

[100] Jorngtzong H, Chiwei C. A mechanism for view consistency in a warehousing system[J]. Journal of Systems and Software, 2001,56(1):3-37.

[101] Wu M C, Buchmann A P. Encoded bitmap indexing for data warehouses[C]. In: Proc. of the 4th International Conference on Data Engineering(ICDE). Los Alamitos:IEEE Computer Society Press,1998. 220-230.

[102] 李盘林等编著. 离散数学[M]. 北京:高等教育出版社. 2000

博士期间论文发表情况

1. 胡孔法,董逸生,徐立臻等. 一种基于维层次编码的 OLAP 聚集查询算法. 计算机研究与发展, 2004, 41(4):608-614
2. 胡孔法,董逸生,徐立臻等.多维数据仓库系统中高性能 DCA-Tree Cube 的研究. 应用科学学报, 2003, 21(2):137-140
3. 胡孔法,董逸生,徐立臻等. 基于 OLAP 查询的数据仓库视图的分割. 应用科学学报, 2003, 21(4):362-366
4. 胡孔法,宋爱波,董逸生等. 数据仓库中实视图聚集函数的增量更新. 东南大学学报(自然科学版), 2002,32(1):11-14 ***(EI 已收录)***
5. 胡孔法,董逸生,徐立臻,杨科华. 基于 OLAP 查询的数据集市系统的研究. 东南大学学报(自然科学版), 2002,32(6):875-878 ***(EI 已收录)***
6. KONG-FA HU , YI-SHENG DONG , LI-ZHEN XU . DCA-TREE: A HIGH PERFORMANCE STRUCTURE FOR INCREMENTAL UPDATE CUBE ON MDDW . Proceedings of 2002 International Conference On Machine Learning and Cybernetics(ICMLC02).Beijing,China,2002.11. 2069-2072 ***(EI、ISTP 已收录)***
7. Hu Kongfa, Dong Yisheng, Xu Lizhen.INCREMENTAL UPDATE CUBE FOR MOLAP RANGE QUERIES ON MDDW. Proceedings of the 8th Joint International Computer Conference (JICC2002).Ningbo,China,2002.11.583-586
8. 胡孔法,董逸生等. 数据仓库中基于实体化辅助视图的视图增量维护. 小型微型计算机系统. 2003,24(2):251-254
9. 胡孔法,董逸生等. OLAP 中聚集函数的更新.《第十八届全国数据库学术会议论文集》,2001
10. 胡孔法,董逸生,徐立臻等.一种基于维层次聚集树的 Cube 增量更新算法. 应用科学学报，已录用.
11. 胡孔法,董逸生,徐立臻等. 基于维层次的压缩 Cube.《东南大学学报》(自然科学版),已录用.
12. 宋爱波,胡孔法,董逸生.Web 日志挖掘. 《东南大学学报》(自然科学版）,2002, 32(1):15-18 ***（EI 已收录）***
13. 宋爱波,胡孔法,董逸生.Weblog 的模糊聚类.《第十八届全国数据库学术会议论文集》，2001

博士期间参与的科研项目

1. 数据仓库系统研究与开发，江苏省“十五”高科技项目（编号：BG2001013）.