Nanjing University of Aeronautics and Astronautics

The Graduate School

College of Information Science and Technology

Research and Implementation of Data Model and Query Language in Spatio-Temporal Databases

A Thesis in

Computer Science and Technology Engineering

by

Duan Hai-Liang

Advised by

Prof. Qin Xiao-Lin

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Engineering

December, 2009

承诺书

本人声明所呈交的硕士学位论文是本人在导师指导下进行的研究工作及取得的研究成果。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，也不包含为南京航空航天大学或其他教育机构的学位或证书而使用过的材料。

本人授权南京航空航天大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

(保密的学位论文在解密后适用本承诺书)

作者签名：段海亮

日　　期：2010.3.19

摘要

时空数据库是在空间数据库、时态数据库基础上形成的研究热点。时空数据库主要用于处理随时间离散或者连续变化的空间对象，其核心内容为时空对象数据建模、存储结构和拓扑分析操作，目的在于为空间信息尤其是时空信息提供一个高可靠、高效率的基础软件平台用于分析和决策。本文主要针对时空数据库研究中的一些关键性问题进行有益和深入的讨论，主要工作包括以下几个部分：

(1) 在讨论时空数据库核心问题的基础上，通过使用基于数据类型的时空数据建模思想，按照从抽象到离散的建模思路，给出了完整的时空数据类型系统定义，并提出两类新的时空数据类型，丰富了原来类型系统的表达能力。第一类是可以支持描述历史-将来运动的时空对象的数据类型，这种类型可以很好的支持全时间域内的查询，解决了之前的数据类型不能很好的同时支持过去和将来运动的缺点。第二类是可以支持时空对象在运动周期内可能发生分形(拓扑结构发生变化)的时空复合类型，这种类型可以表示更复杂的时空对象，解决了之前一种数据类型只能描述一种拓扑结构时空对象的缺点。

(2) 根据提出的时空数据类型系统，给出时空代数系统实现时的核心数据结构，为了同时高效的支持不同类型的查询，对时空对象的快照元素采用顺序和二级平衡二叉树 AVL 两种存储结构，实现时采用 STL 中的 Vector 和 Set 数据结构。在此基础上，共设计实现了 5 大类 92 个时空操作，时空数据类型和这些操作结合在一起形成了一个完整的时空代数系统。最后给出了一个典型时空操作算法的具体实现。

(3) 基于 O-RDBMS AMOSII 扩充实现了时空数据库管理系统 NHSTDB。详细讨论了扩充的技术路线及查询语言的设计，给出 NHSTDB 中时空类型体系和相应的操作函数，设计实现了 9 个 NHSTDB 时空类型数据信息到时空代数系统的操作结构转换算法。最后通过一个实际的应用实例，详细说明了 NHSTDB 的使用方法，证实 NHSTDB 的易用性和高效性。

关键词： 时空数据库，数据模型，复合数据类型，查询语言，NHSTDB

Abstract

Spatio-temporal databases is the research hotspot on the basis of spatial databases and temporal databases, which is mainly used for processing discrete or continuous changes with time in spatial objects, and the cores are spatio-temporal objects data model, storage structure and topological analysis operations, with the aim to provide a highly reliable and efficient software platform for analysis and decision making of spatio-temporal information. Many key issues about spatio-temporal databases are discussed profoundly in this paper, as follows.

On the basis of discussing the core issues of spatio-temporal databases, through the use of spatio-temporal data modeling based on the data type, from the abstract level to the discrete level, a complete spatio-temporal data types system is provided, and two new types are proposed, enriches the expressive power of the original types system. The first type which can support the both past and future movements of spatio-temporal objects, fully support full-time queries, and resolve that the prior data type could not support the past and future movements at same time. The second type spatio-temporal composite type which can support fractal in the movement life of spatio-temporal objects, can represent more complex spatio-temporal objects, and resolve that a data type can only describe a topological object.

According to proposed spatio-temporal data types, the implementation of core data structure is given. In order to support different types of queries effectively, two kinds of storage structures are used which are spatio-temporal snapshots by sequence and two-level balanced binary tree(AVL-tree) at the same time. On this basis, one typical algorithm is presented.

Based on O-RDBMS AMOSII expansion enabling spatio-temporal database management system NHSTDB, technical routes and query language design are dicussed in detail. Finally, a practical application example, demonstrating the use NHSTDB, confirms it is easy and efficiently to use the prototype system.

Key words: Spatio-temporal databases, data models, complex data types, query language, NHSTDB

目录

图清单

表清单

注释表

DBMS	Database Management System	数据库管理系统
O-RDBMS	Object-Relation DBMS	对象关系数据库管理系统
OODBMS	Object-Oriented DBMS	面向对象数据库管理系统
AMOSII	Active Mediator Object System II	主动中介对象系统 II
AMOSQL	AMOS Struct Query Language	AMOS 结构化查询语言
ROSE	Robust Spatial Extensions	健壮的空间扩展
TGIS	Temporal Geographic Information System	时态地理信息系统
STAL	Spatio-Temporal Algebra software package	时空代数软件包

第一章 绪论

随着存储技术、传感器设备、数据库技术和基于位置服务(Location Based Service，LBS)的不断发展和普及应用，采集大量时空数据信息已经成为现实，数据量也呈海量规模增加。因此对时空信息的处理需求日趋加大，对高性能时空数据库的需求也越来越迫切。在民用领域，时空数据库可以应用在 GIS、交通网络、气象预测、抗灾防灾和多媒体等多个方向；在国防领域，时空数据库是指挥自动化系统的核心。由此可见，这项基础软件技术在民用以及军事领域都有很大的应用价值和发展前景，正在成为国内外研究的热点。

1.1 研究背景

过去三十年，数据库管理系统研究和开发都已经取得了巨大成功，当然其中也包含很多特种数据库，比如内存数据库、实时数据库、嵌入式数据库等，也出现了很多知名的数据库厂商如 Oracle，Sybase，Mysql 等。差不多世界上所有主要公司都使用了数据库管理系统，创造了 100 多亿美元的行业价值。对于企业而言，不使用数据库管理系统来管理巨大的数据，简直就是不可想象的[1]，因此对数据库的研究一直是学术界研究的热点。

时空数据库在二十世纪八十年代末开始为人们所重视，G. Langran 在 1992 年撰写了关于时空数据库的第一本专著 Time in Geographic Information System，为时空数据库研究的推进做出了重要贡献，时空数据库现在已经成为数据库领域中备受关注的前沿方向。时空数据库是时态数据库[2]与空间数据库[3]的统一体，即包括时间与空间要素信息。时空数据库的本质是存储和管理位置或者形状随时间而离散或者连续变化的各类空间对象。空间数据库也可看做是时空数据库的一个特例，在空间数据库中所有数据对象的位置或形状不随时间而变化(即有零速度)。

时空数据库涉及到的研究内容相当丰富，主要涉及时空对象表达、时空对象建模、时空对象索引、时空对象查询、时空数据库体现结构等。同时时空数据库原型系统、时空推理、时空查询语言描述、时空查询代价模型、时空信息的图形化用户界面等也为时空数据库的研究带来了一定的挑战[4]。

时空数据库的实现可以使数据库成为真正意义上的资源清单，目前的空间数据库基本上不存储旧的、过时的数据，而时空数据库则包含大量的历史数据，可以对历史、当前和将来进行对比、分析、检测和预报预测，从而为预测预报系统、决策支持系统和其他分析系统服务。

时空数据库在地理信息系统、多媒体应用、智能交通、导航系统、生态环境系统、数字战场等方面具有广泛的应用前景和潜在的经济价值，从而激发了世界上广大科研工作者及有关商家的浓厚兴趣，尤其是欧洲国家已经开展了许多相关的研究工作。虽然有些 DBMS 厂商已经研制了空间数据库系统，对时空数据库也进行了深入的研究，但是还没有提出完全可行的技术方

案和商用的时空数据库原型系统研制出来[5]，所以很有必要对时空数据库继续进行深入的研究。

1.2 时空数据库核心问题

传统的数据库管理系统(DBMS)为时空对象应用提供了基础，但是它并不能完全满足时空对象应用的要求，而且并不能从数据库的底层表示、存储层面和查询操作上对时空信息处理提供良好的支持。因此时空数据库要求在现有的 DBMS 上集成和扩展一系列新的功能。下面分别介绍新的时空对象应用对传统的 DBMS 所提出的一些主要挑战：

1．时空数据的建模与表达

数据建模的主要任务是定义数据类型、操作及其之间的关系，其核心问题就是如何表示和实现时空对象[3]。由于现在的计算机系统并不能直接表示连续的对象，因此只能采取近似的、离散的方法，尽量精确的表示时空对象的运动，同时也要为以后的时空操作和查询分析提供良好的基础，因为不当的方法会增大操作结果的误差。此外，对时空对象的位置变化的表示必须简洁，还要具备较强的表达能力，同时还要易于实现。一个模型的表达能力越强，就能捕捉越复杂的语义，就越接近实际应用，当然数据模型本身也就会越复杂。

时空对象的概念使得传统数据模型中操作的定义增加了新的一维。例如，传统的两个对象间的距离即欧几里德距离，但是在时空对象应用中，距离变成了一个关于时间的函数，而不再是只是一个常数。这就意味着要么对现有的操作进行修改以满足时空对象的应用的要求，要么直接定义新的操作，引入新的数据类型。为了支持时空对象，除了传统数据模型已有的空间和时间的数据类型和数据操作，还需要哪些时空类型、时空操作以及如何高效的实现这些类型的表示和操作，就成为一个非常重要的问题。因为一个完善的数据模型，需要丰富的数据类型，需要更完善、更具表达力的操作，并且要求在定义的操作集上是封闭的。

2．查询处理

现有的查询语言大多数都是非时态的，限于访问单个数据库状态。传统的数据库查询语言（例如 SQL）并没有考虑对象状态的可变特性。在时空对象数据的查询处理中，需要新的空间操作、时态操作和时空操作。这些操作必须适用于不同种类的对象，例如自由的时空对象或者受约束的时空对象。因此问题就在于如何在已有的查询系统基础上设计实现新的查询系统，使其能处理时空对象数据的动态特性。而且，时空对象应用经常会利用不同的数据库回答查询，这就意味着查询处理必须考虑延迟、负载以及误差等性能方面的因素[6]。

由于时空对象的移动特性，在提出查询的时候为真的时空条件随时可能变为假，所以提出的查询需要经常重新处理。因此，什么时候以及如何重新处理查询也成为一个重要的问题。

3．索引

时空数据库需要处理大量的时空信息数据。为了回答时空查询，检索数据库中时空对象就会导致很高的性能开销。因此，需要对时空对象的相关属性尤其与时空相关的属性进行索引以

提高查询速度。然而，直接利用空间索引是不可行的，因为时空对象的变化和相关属性是频繁的，从而导致空间索引频繁更新[7]。所以需要提出新的时空索引技术。之前时空索引的工作或者是针对历史数据，或者针对当前和未来的数据，但是多数的方法处理的都是随时间离散变化的空间对象[8]。因此，如何对连续变化的时空对象数据进行索引，并且提供较好的性能以及可接受的开销，就成为时空数据库研究的一个重要问题。

4. 不确定性和不精确性

不精确性是由于时空对象在数据库中存储的位置和其实际位置的差异引起的，是数据采集设备的误差带来的；不确定性是时空对象的位置属性本身所固有的，是其运动过程中固有。两者的存在给数据库建模、查询和索引带来了很多复杂的问题。时空对象的记录越精确，就需要越频繁的数据更新，查询结果也就越准确，却会导致较差的查询性能和较大的存储代价。因此，时空数据模型必须在允许的一定的误差的前提下表示时空对象，同时又不能对查询性能造成太大影响，这里就需要在精确性和效率两者之间找到平衡[9]。误差处理的一个主要的研究问题就是，如何处理时空对象数据固有的误差以及相关的时空查询分析。

5. 高效的存储机制

时空对象应用中有很多对象需要跟踪，跟踪过程中采集数据的频率很高，从而产生大量的数据信息，因此需要高效的存储机制能支持时空操作。一方面，并不是采集到的所有数据都是必要的；另一方面，需要研究更适合的数据采集策略，收集关键的数据，使用高效的更新策略。为了节约存储空间，有时需要使用一些数据压缩技术，比如采用冗余数据删除技术。

1.3 课题研究的目标

本课题的目标就是在课题组之前研究的基础上给出一种通用的、具有较强表示能力的基于数据类型的数据模型。这种模型不能像早先那样面向应用建模，必须从数据库底层的角度出发进行时空数据建模，使数据库本身而不是上层的应用程序来支持各种时空数据类型及其时空分析操作。并给出在此模型之上的正确的、完备的、健壮的时空分析操作。基于该数据模型给出一种兼容 SQL3 的时空数据查询语言，采用与 STSQL 类似的设计方法。课题并不是从零开始设计一个新的时空查询语言，而是适当的扩充广泛使用的 SQL，课题主要关注集成开发。最后集成实现一个时空数据库管理系统 NHSTDB，该系统可以比较高效的实现时空数据信息的表示、存储、管理与处理。并通过实例证实 NHSTDB 的实用性和高效性。

1.4 本文的研究工作和组织

1.4.1 主要研究工作

本课题来源于国家 863 高技术研究发展计划项目——基于网格的数据可靠存储与容侵关键技术。本文主要工作是建立在课题组已有的研究基础之上，设计实现了基于数据类型的时空数

据模型；基于可扩充对象数据库 AMOSII，研究和设计时空数据库原型系统 NHSTDB。

1.4.2 本文的组织和安排

本论文共分为 6 个章节，深入讨论了时空数据库研究和实现中面临的几个关键技术。

第 1 章为“绪论”。介绍论文的来源。时空数据库的现状及其发展，重点分析了时空数据库的几个核心问题。并给出了本论文研究内容、论文的组织结构和研究目标。

第 2 章为“时空数据库概述”。首先简要说明了时空数据库现有的数据模型，然后详细介绍了时空查询语言、时空数据库的索引技术，最后还讨论了时空数据库管理系统的实现方案。

第 3 章为“基于数据类型的时空数据模型”。根据抽象-离散数据类型系统设计思想，讨论了时空数据类型的类型系统和所需要的数据类型。为了描述更复杂的时空对象，提出了两类复合时空数据类型，其一是支持在运动过程中形状发生分形(拓扑类型发生变化)的时空对象的复合数据类型；另外一种是可以同时支持过去和将来运动的时空复合数据类型。

第 4 章为“时空数据库的查询语言”。按照第三章给出的数据类型系统，详细讨论该类型系统中最核心的数据结构的设计，最后给出了典型的时空数据操作的算法实现。这样，时空数据类型和时空操作形成了支持时空查询的语言。

第 5 章为“时空数据库管理系统 NHSTDB 设计与实现”。通过扩充 AMOSII，设计实现了一个原型系统 NHSTDB，使其支持时空数据类型和操作。给出了 NHSTDB 的内存布局图，体系结构，时空对象存储方案等，最后给出了 NHSTDB 的一个应用实例。

第 6 章为“总结和展望”。总结了本文的主要工作，对研究中的不足和有待完善的地方进行了探讨，对时空数据库的发展进行了展望，并在最后给出了课题研究过程中对时空数据库的一点思考。

第二章 时空数据库概述

2.1 时空数据模型

时空数据模型是描述现实世界中的时空对象、时空对象间的时空联系以及语义约束的模型。自20世纪90年代开始，学者已提出了多种不同的时空数据模型[10]。大致可分为5类：基于版本的时空数据模型、基于事件的时空数据模型、基于约束数据库的时空数据模型、基于数据类型的时空数据模型、面向移动对象的时空数据模型。下面主要讲述其中比较经典的几类。

1. 基于约束数据库的时空数据模型

约束数据库[11][12](constraint databases)是20世纪90年代中期提出的一种新的数据库技术，其理论基础是约束数据模型。约束数据模型通过“广义元组（generalized tuple）”对传统的关系数据模型进行了扩充。约束关系R中的一个约束元组是定义在一个变量集上的约束合取式。例如，定义在变量集{x,y}上的约束元组可以是“((1<x<3)∧(2<y<5))”，这个约束元组对应于二维平面上的一个矩形区域。由于每个约束元组可以描述一个可能是无限的点集，因此约束数据模型可以用约束的形式来表示时空数据等多维信息。

基于约束数据库的时空模型[13][14][15]中，空间对象采用一个约束集来表示，时空对象的方向、速度、轨迹和曲率等看做对象的重要属性，因此这些属性可以用作约束。在该模型中，一个时间上的线性约束集合表示时间区间，一个从时间维映射到多维空间的线性函数表示空间对象的位置，而一个时空对象运动轨迹的片段则建模为一个约束集、时间区间和坐标变量。

基于约束数据库的时空数据模型最大优点是其在传统关系代数操作上的封闭性很容易证明。因而这种时空数据模型仍可以采用关系代数或者关系演算作为数据操作。这种模型可以很方便的表达连续型变化的时空对象。但是这类模型的主要问题是在实际应用中必须事先知道定义在时间域上的时空对象的约束表达式，而这在实际情况中是很难做到的。

2. 基于版本的时空数据模型

版本是时态数据库中的核心技术之一。G. Langran 在1988年首先提出了在GIS中引入时态信息的时态GIS(TGIS， temporal GIS)概念。通过记录空间对象在不同时间里的状态来记录空间数据随时间而发生的变化，这就是基于版本的时空数据模型的核心。与时态数据库类似，在时空数据库中，空间对象随时间而发生的变化也可采用不同的版本技术。基于版本技术，研究者提出了5中时空数据模型，分别是时空快照模型、基态修正模型、时空立方体模型、时空复合模型、时空对象模型等。时空快照模型以一系列的数据库快照来表示空间对象随时间而发生的演变，其中每个快照记录了当前时刻的数据库状态。这一模型非常简单和直观，但无法表达两个快照间的变化。

3．基于数据类型的时空数据模型

早在1987年，空间数据类型就已经在空间数据库中广泛地应用。这些空间数据类型不仅有严密的形式化定义，而且也能方便的用在查询语言中，甚至有些原型系统是试图将这些空间数据类型集成到数据库中。然而，一直没有一个令人满意的完整的解决方案。原因是由于当时的模型和原型系统都没有数据有限表示的方法。为了解决这个问题，R.H.Güting教授和Schneider教授提出了realm[16]的概念，realm是用户定义的、建立在离散网格上的有限的点和非相交的线段集合构成的结构，用来作为其它数据类型的基础。之后，他们在realm概念的基础上定义并实现了一个代数系统，即ROSE代数系统（Robust Spatial Extension Algebra）[17][18]。在ROSE代数中，所有的空间数据都由建立在realm上的点、线、区域组成，基于realm的ROSE代数能有效的表示空间数据和支持复杂的空间分析操作，并能保证数值健壮性和拓扑正确性。1996年，R.H.Güting在可扩充的DBMS-O2调用ROSE代数；瑞典皇家理工学院等联合在O-RDMBS——AMOS上开发了基于ROSE代数的O-R空间分析DBMS——ORSA[19][20]；我所在的课题组研制的空间数据库NHSpatial[21]也是选用realm作为空间对象的表示模型。

基于ROSE代数系统，1998年Erwig等人提出了将时空对象封装成抽象数据类型ADT（Abstract Data Types）[22]，从而能够集成到数据库管理系统中。该方法的思想是首先提出非时态数据类型的代数系统，然后对其进行扩展提升。使用基于数据类型方法的优点是可以和对象关系数据库管理系统[23]（object-relational DBMS，O-RDBMS）以及SQL语言无缝结合。O-RDBMS提供了数据类型和操作的扩展能力，而且扩展的类型和操作可以直接在SQL中使用。

Parent等人在1999年提出了基于数据类型在概念层次上建模的方法MADS[24]，MADS支持静态对象和移动对象的建模，能同时表达随时间的离散和连续的几何变化。

R.H.Güting教授在2000年提出了时空对象的抽象模型[25]，采用属性数据类型表示时空对象，并为数据类型提供相应的数据操作。这些可以为现有的数据管理系统的数据模型和查询语言提供一个抽象时空类型扩展包。该方法非常符合客观世界。

通过研究总结现有的时空数据模型，得出当前时空数据模型主要的问题有以下几点：

(1)表示能力有限。一般时态GIS中采用的建模方法，如基于事件的时空数据模型，只能表示离散变化，比如地籍的变化，国家版图的变更等，无法表示火灾的蔓延、移动车辆轨迹的变化和台风的运动等连续变化，而且现有的模型基本上没有提供更加复杂而实用的数据类型；

(2)不适合进行复杂拓扑分析。如基于栅格的各类型时空模型、矢量的基态修正模型、时空复合体模型等不能用于复杂时空对象之间的拓扑分析；

(3)通用性低。大多数模型是基于商业GIS系统的时态扩展，或者用于科学研究的原型系统和针对特殊应用的时空查询工具，基本上是针对特定的应用而设计的，只能使用特定的数据结构，通用性非常弱。而且对用户的要求比较高，需要属性很多内置数据类型的格式要求。

2.2 时空数据库查询语言

时空数据库查询语言与时空数据模型是密不可分的。每种时空数据模型总是伴随着属于自己的查询语言。以往提出的时空数据库语言主要有两大研究方向：基于 SQL 的时空数据库语言和基于 OQL 的时空数据库语言，其中基于 SQL 的时空数据库语言又分为 STSQL 和 STQL。SQL 语言是流行的关系数据库语言，已经得到了包括 Oracle、IBM 等主流数据厂商的支持。OQL 是 ODMG(Object Data Management Group, 对象数据管理组织)提出的面向对象数据库（OODBMS, Object-Oriented Database System）查询语言，已经成为面向对象数据库的标准语言，并得到了不少 OODBMS 的支持。

基于 SQL 的时空数据库语言 STSQL[26]是基于数据类型的时空数据库所采用的查询语言。由于 SQL3 提供了抽象数据类型的扩展能力，因此基于数据类型的时空数据模型能够与 SQL3 无缝地结合。STSQL 的特点是兼容 SQL 语言。它没有扩展 SQL 语言的子句，只是扩展了数据类型和操作。从这个意义上说，STSQL 具有较强的可应用性，因为基于 SQL 的数据库和 SQL 语言目前在数据库应用中占据主导地位。

STQL 是另一种基于 SQL 的时空数据库查询语言[27]。STQL 的时空数据模型以关系模型为基础，并以元组版本的形式来表示时空数据和时空变化。但由于 STQL 扩展了 SQL 的子句，因此与 SQL3 是不兼容的。

STQL 的数据查询语句 SELECT 子句通过两种方式对传统的 SQL 查询语句进行了扩展：一方面引入 WHEN 子句表达时态上的查询条件，另一方面引入空间操作和空间谓词来表达空间数据上的查询条件。因此，STQL 实际上是通过时态条件和空间条件的组合来表达时空查询的。与 STSQL 相比，STQL 人为割裂了时空查询，而且由于采用的基于版本的时空数据模型，因此无法表达连续变化查询。

还有一些以 ODMG 对象模型为基础的时空数据模型和时空数据库查询语言。面向对象的数据模型提供了比关系模型更强的表达能力，因此在表达时空数据和时空变化上比其它方法更具优势。采用面向对象模型，时空数据可以用对象的方式来表示，并且时空操作也可以用对象上的操作来表达。文献[28]介绍了一个面向对象的时空数据库系统，它采用了 OQL 的时空查询语言，典型的时空数据库查询示例“查询某个时间区间 Range 内的时空对象”如下所示：

```
SELECT c
FROM c IN Spatio_Temporal_Objects
WHERE c.overlap(range)=true;
```

面向对象数据库提供了高度的数据抽象和建模能力，非常适合表达复杂的时空数据。但是由于面向对象数据库中存在一些问题还没有解决，使得面向对象的时空数据模型在实现和实际应用上存在一定的障碍。

因此，综合上述因素，从目前的现状来看，基于 SQL 的时空数据库语言 STSQL 是时空查询语言研究的主流，尤其是随着对象关系数据库技术的不断发展和成熟，结合 SQL3 和对象关系数据库技术的时空数据库系统越来越受到了研究人员的关注。

2.3 时空数据库索引技术

时空数据是多维数据，选择高效的存取方法，提高时空多维数据的存取效率，也是时空数据库的重要研究内容之一。实际上，从时空数据库的初始研究阶段起，就开展了对索引技术的研究。由于查询是数据库应用中的主要操作，因此尽管索引技术带来了索引建立和更新等额外代价，但对于应用的总体效率而言仍是值得。目前针对时空数据的索引技术中，还没有一个能同时支持过去和将来运动的时空索引。

目前，主要的时空索引结构有四种，分别为 3DR-trees[29]、RT-trees[30]、MR-trees[31]和 HR-trees[32]。3DR-trees 将时间看成空间的另一维，当时空对象在一段时间内保持静止时形成一个立方体；RT-trees 按照时空对象的空间分布组织索引，在每个节点同时放上对象的空间数据信息和时间区间。当对象的空间信息发生改变时，产生一个新的数据项插入到索引中。因此，RT-trees 是一个包含了时态信息的空间索引；Mr-trees 和 HR-trees 采用 R-trees[33]形式组织每一个时刻的空间信息，它保存了不同时刻时空对象的空间分布，实现时采用子树重叠。HR-trees 单独处理时间维，先将时间戳建立索引，然后建立对应每个时间戳的空间索引数据，也就是先索引时间再索引空间。使用 HR-trees 进行时间片查询时，先要找到对应时刻 t 的 R-trees 根节点，然后进行空间查询。因此 HR-trees 能很好地支持时间片查询，但是对时间段查询却比较低效，因为同一数据项可能被多次检索，数据冗余大，空间开销太大。所以需要设计一个很好支持时间片和时间段的时空查询的索引。

2.4 时空数据库实现结构

时空数据库管理系统实现结构的研究继承了空间数据库和时态数据库的研究成果。目前已提出的实现结构主要有三种：完全型时空数据库实现结构[34]、层次型时空数据库实现结构[35]和扩展型时空数据库实现结构[36]，文献[37]对这三种体系结构进行了详细的讨论和对比，给出了它们的优缺点和下一步研究目标。

1．完全型时空数据库实现结构

完全型时空数据库实现结构直接在操作系统的基础上实现一个时空数据库管理系统。这种结构需要完全实现一个数据库管理系统中的模块，包括查询编译与执行、事务管理、存储管理等，并且为了满足实际应用，还要设计时空数据库的数据驱动程序。这种结构的实现工作量非常大、设计非常复杂，而且难以满足一般时空应用开发的需求，不适合个人研究和开发。要想实现基于完全型时空数据库实现结构的原型系统，需要 DBMS 厂商和研究机构进行深入的合作

研究和开发。

2. 层次型时空数据库实现结构

如图 2.1 所示，层次型时空数据库实现结构在传统的关系数据库管理系统之上添加了时空处理层，通过时空处理层来完成对时空数据的操作，不需要对底层的数据库管理系统内核作任何修改。时空处理层负责时空数据库语言与 SQL 间的翻译、时空查询优化等工作，所有的时空数据请求都要通过时空层进行处理。时空查询转换成的 SQL 十分复杂，不利于底层的关系数据库管理系统进行查询优化。另外，时空层会成为应用开发的瓶颈，因为所有的请求都要首先通过时空处理层转换为标准的 SQL。这种体系结构的原型系统的时空操作效率将主要取决于时空处理层。

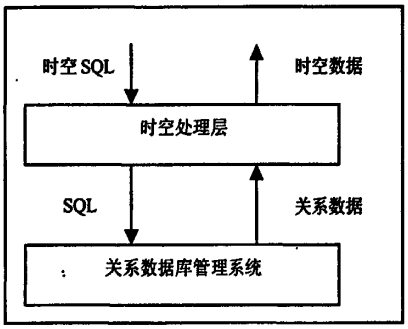

图 2.1 层次型时空数据库实现结构

3. 扩展型时空数据库实现结构

扩展型时空数据库实现结构是在对象关系数据库管理系统上进行时空扩展。图 2.2 表示了这种结构，对象关系管理系统内核中部分内容是时空扩展。由于对象关系数据库管理系统提供了用户定义数据类型(User Defined data Type,UDT)和用户定义过程(User Defined Routine,UDR)的扩展功能[23]，因此，可以在对象关系数据库管理系统的基础上扩展新的时空数据类型和时空操作，并将时空索引也通过 UDR 和其他技术扩展到 DBMS 内核中。这种实现结构可行性比较强，已经受到研究人员的关注。但它也存在一些问题，虽然底层的数据库管理系统可以使用扩展的时空索引来加快查询速度，但它对时空查询优化的支持也仅局限于此。底层的数据库管理系统仍然采用关系数据库的查询优化规则来处理时空查询，显然这并不适于时空查询。

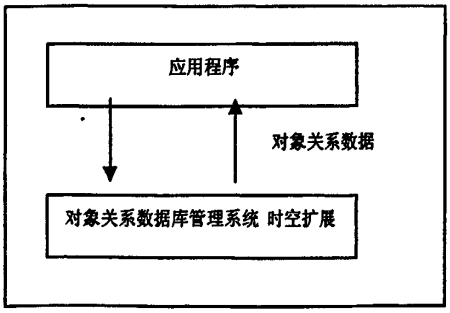

图 2.2 扩展型时空数据库实现结构

2.5 本章小结

本章对时空数据库进行了概述，首先介绍了时空数据库数据模型，主要讨论了基于约束数据库的时空数据模型、基于版本的时空数据模型和基于数据类型的时空数据模型，讨论了时空数据库的查询语言和索引技术，最后讨论了时空数据库管理系统的三种设计方案，包括完全型，层次型，扩展型。本章为本文进一步对时空数据模型和系统实现技术的研究工作打下了良好的理论基础。

第三章 基于数据类型的时空数据模型

3.1 引言

传统的数据库管理系统只给用户提供数量固定的类型集合，而且这些都是简单的类型，不足以管理信息量相对较大的空间或者时空数据信息。现有的表示方法也只是在应用层做的处理，没有从数据库的底层对空间或时空数据类型给予支持。在空间数据库研究中，为了满足空间信息存储的需要，研究人员采用扩展数据库中基本类型的方案，引入了空间数据代数系统。定义了能表示空间对象的数据类型和相关的操作，在关系或者其他数据库系统中扩充使用。在许多空间查询语言中已经使用了空间类型和操作，并已经实现了一些数据库原型系统。同样，为了描述和存储时空数据，完全可以采取类似的方案，在空间代数系统的基础上，扩展基本数据类型和空间数据类型，定义专门的代数系统实现时空类型和操作，即时空代数系统。

定义一个代数系统包含两个步骤：首先，通过引入一些基本类型和类型构造符，设计一个类型系统，通过定义类型的载体集来表达类型系统中每种类型的语义；然后，为类型系统中的每种类型设计相关的操作集合，通过定义标记签名(signature)描述每种操作的语法，通过定义参数类型载体集上的函数给出操作的具体语义。

本章采用基于数据类型的类型系统建模[38][39]的方法，从抽象和离散两个层次对时空类型的进行了定义。本章前半部分完整地定义了时空数据库类型系统，基于无限集建立各类空间对象、时空对象的抽象模型，规定各类数据的表示规则，用于指导离散实现模型的设计。然后在抽象模型的基础上，提出相应的离散模型并给出数据类型的离散定义，使用基于有限集的离散模型来提供对数据类型和数据操作(尤其是时空拓扑分析操作)的支持。最后提出了两类复合时空数据类型，来表达更加复杂、更加实用的时空对象。

3.2 类型系统

采用参考文献[38]中介绍的类型系统建模的方法，首先给出 NHSTDB 中时空模型的类型系统。如表 3.1 所示。

表 3.1 时空数据类型系统

类别 ： BASE，TIME，SPATIAL，RANGE，TEMPORAL，HF ，CFMDT	
类型构造符	标记
int,real,string,bool	→BASE
point ,points,lines,regions	→SPATIAL
instant	→TIME
moving ,intime	BASE∪SPATIAL →TEMPORAL
range	BASE∪TIME →RANGE
hf	TEMPORAL →HF
cfmdt	TEMPORAL →CFMDT

以上是模型的类型系统名称定义，下一节会详细介绍它们的语义和载体集。类别 BASE 中的常量类型 int、real、string、bool 都是基本数据类型，这些类型是普通的数据库管理系统具有的，只需对其定义域做简单的修改。类别 SPATIAL 中的常量类型 point、points、lines、regions 是空间数据类型，由空间数据库提供。类别 TIME 中的常量类型 instant 是基本的时间类型，可以将其视为一个实数或者是一个附加了若干操作的时间类，在研究中暂时用了实数简单表示。类型构造子 range 可作用于类别 BASE 和 TIME 中的所有类型，从而产生 range(int)、range(real)、range(string)、range(bool)、range(instant)。类型构造子 moving 可作用于类别 BASE 和 SPATIAL 中的所有类型，从而产生新的类型，例如 moving(points)表示移动点类型，moving(real)表示随时间变化的实数，可以作为一些时空操作的返回类型。类型构造子 intime 可作用于类别 BASE 和 SPATIAL 中的所有类型，从而产生新的类型，例如 intime(points)类型是一个二元组，表示在某个时刻时空点对象的位置。类型构造子 hf 可作用于 TEMPORAL 类型，从而产生可以支持过去和将来运动的复合时空类型。如 hf(mpoints,mlines)表示一个在目前时刻之前的运动为移动点，之后预期为移动线的时空对象。类型构造子 cfmdt 可作用于 TEMPORAL 类型，从而产生可以支持在运动发生拓扑变化的时空数据类型。

从上面的分析可见，虽然时空数据库关注的时空对象，即主要是移动点、移动线、移动区域类型，但是为了保证类型系统的完整性，类型系统必须包含相关的其他数据类型，例如基本数据类型、时间数据类型、空间数据类型以及时态数据类型 mreal 等，这些辅助的类型是时空操作过程中要用得到得。

3.3 抽象数据类型

3.3.1 基本数据类型

基本数据类型有 int、real、string 和 bool，这些类型与传统的定义相同，只是对每个类型的载体集作了扩展，增加了⊥值(undefined)，因为在时空查询中很多操作结果在某个时刻或者时间段内可能为未定义值。对类型 α，A_α表示 α 的载体集。

定义 3.1 类型 int、real、string 和 bool 的载体集的定义如下：

$A_{int}=Z\cup\{\perp\}$，其中，Z 是整数

$A_{real}=R\cup\{\perp\}$，其中，R 是实数

$A_{string}=V^*\cup\{\perp\}$，其中，V 是有限的字符表，

$A_{bool}=\{FALSE, TRUE\}\cup\{\perp\}$，

$\overline{A_\alpha}=A_\alpha-\{\perp\}$。

3.3.2 时间数据类型

Instant 类型表示时间上的一个点或未定义。在 NHSTDB 的时空模型中，时间被看成是连续

的，相当于实数。

定义 3.2 instant 类型的载体集定义为：

$A_{instant}=R\cup\{\perp\}$，其中，R 是整数。

3.3.3 空间数据类型

空间数据库中，基本空间概念有点、线和区域。点表示仅和位置而与范围无关的对象，如公交站点；线是对平面上的移动路径或平面上的连线的抽象，如高速公路、飞机航线；区域是对那些与位置、范围都相关的空间对象的抽象，如湖泊，城市。

在类型系统中，空间数据类型有：point、points、lines 和 regions，如图 3.1 所示。

空间数据类型的形式化定义是基于无限点集理论和点集拓扑理论。无限点集理论认为，空间是由无限的点组成的，各个空间对象作为实体的区别在于它们是这个无限点集的不同子集。点集拓扑理论提出了连续性和封闭性的概念，从而可以识别点集的拓扑关系，例如界内 interior、边界 boundary、界外 exterior。

图 3.1 空间数据类型

1．point 和 points 数据类型

point 类型的值表示欧式空间中的一个点或未定义值，points 类型的值表示有限的点的集合。

定义 3.3 抽象类型 point 和 points 的承载集定义：

$A_{point}=R^2\cup\{\perp\}$，其中，R 是实数，

$A_{points}=\{P\subset A_{point}\mid P$ 是有限点集合$\}$。

2．lines 数据类型

一个 line 类型的值表示平面上一个连续曲线 curves 的集合，定义如下：

定义 3.4 curves 是一个映射 $f:[0,1]\to R^2$，且

$\forall a,b\in[0,1]: f(a)=f(b)\Rightarrow a=b\vee\{a,b\}=\{0,1\}$。

令 $rng(f)=\{p\in R^2\mid\exists a\in[0,1]; f(a)=p\}$，给定的两条曲线 f 和 g，若 rng(f)=rng(g)，则 f=g。点 f(0)和 f(1)称为 f 的两个断点。若 f(0)=f(1)=p，则称 f 在点 p 处闭合。

这个定义是允许存在回路的（由于 f(0)=f(1)），但是不允许内部的两个不同点相等或内部点与两个端点中任意一个相等。

定义 3.5 lines 数据类型的承载集定义如下：

$A_{lines} = \{Q \subseteq R^2 \mid \exists f \in CF : Q = rng(f)\}$，其中 CF 表示定义 3.4 中所有映射的集合。

3．Regions 数据类型

若一个集合 Q 的闭包与它本身相吻合，即 Q =closure(interior(Q))，则 Q 为正则闭集。该正则化处理过程的目的是考虑到区域应当是规则的。

定义 3.6 regions 类型的 r 是一个正则闭集，且 $\exists C \subset CF : \partial r = point s(C)$。其中，$\partial r$ 表示 r 的边界。

定义 3.7 $A_{regions} = \{Q \subseteq R^2 \mid \exists R \in RC : Q = point s(R)\}$，其中，RC 表示定义 3.6 中所有正则闭集的集合。

3.3.4 范围数据类型

对于所有的移动类型，有一些投影到函数的定义域或值域的操作。对于与基本类型相对应的移动类型，例如 moving(real)，其在定义域上的投影则可能是一维空间上的区间集。因此，需要引入新的数据类型来表示 real 和 int 等之上的区间集。这里，通过 range 构造子来产生这些新的数据类型。范围类型定义如下。

定义 3.8 令 α 为 range 类型构造子可作用的数据类型，且在 α 上有一个全序<存在，一个 α -interval 是一个集合 $X \subseteq \overline{A_\alpha}$，其中 $\forall x, y \in X, \forall z \in A_\alpha : x < z < y \Rightarrow z \in X$。

若两个 α -interval 相离(disjoint)，且能够合并成一个 α -interval，则称这两个 α -interval 相邻(adjacent)。一个 α -interval 是有限个相离但是不相邻的区间的集合。对于一个 α -interval 类型的 R，可用 points(R)表示其所有区间的并集。

定义 3.9 令 α 为 range 类型构造子可作用的一个数据类型，则 range(α)的承载集定义为：

$A_{range}(\alpha) = \{x \subseteq \overline{A\alpha} \mid \exists 一 \alpha - range型的 R : x = point s(R)\}$。

3.3.5 时态和时空数据类型

类型构造子 moving 把基本类型和空间类型构造出相应的移动类型。任意给定的类型 α，moving 构造子产生一个从时间到 α 的映射。更精确的描述如下定义：

定义 3.10 令 α 为 moving 构造子所能作用的数据类型，其承载集为 A_α (在之前的几个定义中已经对相应的基本类型和空间类型的承载集进行了定义)，则 moving(α)的承载集定义如下：

$A_{moving(\alpha)} = \{f \mid f : \overline{A_{ins}}_{\tan t} \to \overline{A_\alpha}\}$，其中 f 是一个偏函数并且 $\Gamma(f)$ 是有限的。

由定义可知，moving(α)的承载集中的每个值 f 都表示一个 α 类型的承载集中的值随时间变化的函数。条件“$\Gamma(f)$ 是有限的”指的是 f 由有限的连续部分组成。这个条件是必须的，理由是：(i)时空对象在空间上的投影只能由有限的连续部分组成；(ii)便于抽象层模型的实现。

对于所有的时空类型，在参数类型名前加上前缀“m”作为类型的名字，分别是 mint、mreal、mstring 、mbool 、mpoints、mlines 和 mregions。

通过 moving 构造子得到的数据类型都是函数或无穷的<时刻,值>对组成的集合。为了表示该函数中的某个时刻值，即<时刻,值>对，需要引入一种新的数据类型。类型构造子 intime 将一个给定类型 α 转化为一个将时间与值关联起来的类型。

定义 3.11 令 α 为 intime 可作用其上的数据类型，其承载集为 A_α，则 intime(α)的承载集定义如下：

$$A_{intime(\alpha)} = A_{instant} \times A_\alpha$$

3.4 时空数据的离散模型

在本章的 3.3 节中，已经介绍了抽象模型的定义。抽象模型是以无限集为基础的，没有考虑这些集合的有限表示，但是问题在于现有的计算机中无法存储、操作无限集合。因为只有有限集合才能在计算机中存储，因此需要在离散层次上，对抽象模型进行离散化，即离散模型。后面几小节将在抽象模型的基础上，给出相应的离散模型和数据类型的离散定义。

3.4.1 基本类型和时间类型

基本类型的离散定义可利用编程语言中的类型，而时间类型 Instant 则采用 real 类型。

定义 3.12 若 α 为一种数据类型，用 D_α 表示 α 的值域。则有

$D_{int} = int \cup \{\perp\}$

$D_{string} = string \cup \{\perp\}$

$D_{bool} = bool \cup \{\perp\}$

$D_{real} = real \cup \{\perp\}$

$D_{instant} = Instant \cup \{\perp\}$

其中，⊥表示未定义值。有时候不需要带⊥值，因此定义 $\overline{D_\alpha} = D_\alpha / \{\perp\}$。

3.4.2 空间类型

在二维空间中，一个点通常用坐标对(x,y)表示。令 Point=real×real，则

定义 3.13 $D_{point} = Point \cup \{\perp\}$

给定任意两个点 p，q∈Point，$p<q \Leftrightarrow (p.x<q.x) \vee (p.x=q.x \wedge p.y<q.y)$

点集 points 是点 point 的集合，因此有

定义 3.14 $D_{points} = \{p \in D_{point}\}$

Lines 类型在离散层次采用折线 Polyline 集合表示。

定义 3.15 线段集合 Seg 定义如下：

$Seg=\{(u,v) | u,v \in Point, u < v\}$

定义 3.16 Lines 承载集定义如下：

$$D_{line} = \{S \subset Seg \mid \forall s,t \in Seg : s \neq t \wedge collinear(s,t) \Rightarrow disjoint(s,t)\}$$

对 Lines 中的 Seg 集合的要求确保了没有共线但不相离的线段存在，这样就保证了 Lines 中线段的单一表示，因为共线且有重叠部分的线段可以合并成一个线段。

一个 regions 值在离散层次本质上就是一些可以有空洞的闭合的折线段集合，正式的定义是基于 cycle 和 faces 概念[38]，这里不再累述。

3.4.3 范围类型

范围类型主要由两个类型构造子 Interval 和 Range 产生，Range 将给定的数据类型 α（α ∈ BASE∪TIME，在 α 上必须存在全序关系。)转换为基于 α 有限的区间集合。下面给出 Interval(α) 和 Range(α)定义。

定义 3.17 令（S，<）为一个具有全序的集合，一个基于 S 的区间定义：

$$Interval(S) = \{(s,e,lc,rc) \mid s,e \in S, lc,rc \in bool, s \leq e, (s=e) \Rightarrow (lc = rc = true)\}$$

从上面定义可知，一个区间由两个端点 s、e 以及两个闭合标志的 lc，rc 组成的四元组。比如 Interval(time)=(1,5,true,false)，表示一个从时间点 1 到时间点 5 的左闭右开的时间区间。

定义 3.18 range 构造子定义为：

$$D_{range}(a) = \bigcup_1^n interval(a)_i \quad (\forall a \in BASE \cup TIME)$$

其中 $\forall interval(a)_i$ 都不相邻接或者相交，即它们都是互相独立的区间。

3.4.4 时态和时空类型

Intime 表示时间和值的二元有序对，定义为：

$$D_{intime(a)} = D_{instant} \times Da \quad (\forall a \in BASE \cup SPATIAL)$$

单元类型作为时空对象快照表示的一个基础，定义如下：

$$D_{unit(a)} = \begin{cases} D_{interval(instant)} \times D_a & if\ a \in \{bool, int\} \\ D_{interval(instant)} \times \{b,c,d,r) \mid b,c,d \in a, r \in bool\} & if\ a \in real \\ D_{interval(instant)} \times D_a \times D_a & if\ a \in \{point s, lines, regions\} \end{cases}$$

为了在离散层上表示时空类型(移动对象)，引入时空对象快照类型 Snapshot，快照就是单元类型的集合。

$$Snapshot(a) = \bigcup_1^n unit(a)_i \quad (a \in BASE \cup SPATIAL)$$

在快照基础上，离散层次上的时空类型定义如下：

$$D_{moving(a)} = snapshot(a) \quad (\forall a \in BASE \cup SPATIAL)$$

通过上述的定义就可以表示 mreal，mint，mpoints，mlines，mregions 等类型。

3.5 复合时空数据类型

本小节中将重点讨论对前面所述的时空数据模型的扩充和完善，在此基础上提出两类新的

时空数据类型，以支持全时间域(过去时间和将来时间内)的数据类型和运动过程中可能发生拓扑类型分化的数据类型。这两类新的时空数据类型的引入，使得 NHSTDB 的时空类型系统有更强的表达能力，能表示更加复杂和现实的时空对象。

3.5.1 支持历史和将来运动的数据类型

之前对时空数据库的研究中，已经有学者提出了不少的时空对象数据模型和查询语言。这些模型中有的提供了支持时空对象历史运动的数据类型，有的提供了支持时空对象将来运动的数据类型[40][41][42]。截止目前还没有一种完善的数据模型可以提供同时能描述历史以及将来运动的数据类型，因此也就无法支持从过去一直扩展到将来某个时间的时空查询。

可以以气球的形状来为此类时空对象建模，气球的尾端牵引线部分可以想象为时空对象的过去运动，而气球本身可以比拟为时空对象将来运动可能运动的轨迹。例如，飓风眼的过去和预期的将来运动就可以看做一个有牵引线的气球，其中飓风眼过去的运动可以看做是一个移动点的运动，形成一个空间域中的线或者折线，这部分就相当于气球的牵引线部分；飓风眼在将来某个时刻的位置可能在一个不确定区域的任意位置，可以将其视为位置不确定的移动区域 mregions，这部分就相当于气球的本身球体部分。气球牵引线和球体连接点就可以表示一个移动时空对象的目前状态。

通过使用上面的模型，引入新的数据类型表示不同类型的气球状的时空对象，这些类型可以同时支持时空对象历史和将来运动。还给出了几个典型的涉及这类数据类型的操作和查询，将这些新的数据类型简称为 HFDT(Historical-Future Data Type)。

定义 3.19： $time_h=(-\infty, tp]$，$time_f=(tp, \infty]$。其中 tp 表示对象目前状态时间，$time_h$ 为时空对象历史运动的时间域，$time_f$ 为时空对象将来运动的时间域。

定义 3.20：令 $hf(\alpha,\beta)$=moving(α)×moving(β)×MC(β)，满足以下四个条件：(1)dim(β)≥dim(α)；(dim 返回空间类型的维数)。(2)moving(α)表示时空对象过去运动，并且在 $time_h$ 上有定义；(3) moving(β)表示时空对象将来运动，并且在 $time_f$ 上有定义。(4)MC(β)是定义在 β 上的一个移动置信分布，满足只要 MC(β)有定义，moving(β)就有定义。(α,β 可能是空间数据类型 points，lines，regions 中的任意一种)。

由定义 3.20 可知，$hf(\alpha,\beta)$可以刻画 HFDT 类型对象，其中第一个约束条件保证了时空对象在整个运动中维数不会降低。通过细化 α 和 β 空间对象类型，可以得到 6 种 HFDT 数据类型。如下所示：

Hf_pp =hf(points,points) = mpoints×mpoints×MC(points)

Hf_pl = hf (points,lines) = mpoints×mlines×MC(lines)

Hf_pr = hf (points,regions) = mpoints×mregions×MC(regions)

Hf_ll = hf (lines,lines) = mlines×mlines×MC(lines)

Hf_lr = hf (lines,regions) = mlines×mregions×MC(regions)

Hf_rr = hf (regions,regions) = mregions×mregions×MC(regions)

图 3.2 中(a),(b),(c)分别表示 Hf_pp，Hf_pl，Hf_pr 类型的对象实例。

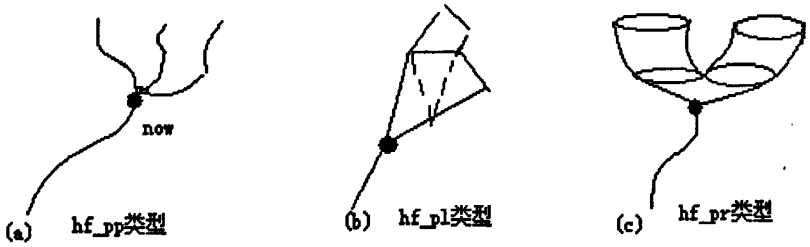

图 3.2 过去和将来运动的时空对象

表 3.2 简要的给出一些关于 HFDT 数据类型的典型操作：

表 3.2 HFDT 类型典型操作

操作	标记	语义
Past	hf(α,β) →moving(α)	查询对象的历史运动
Future	hf(α,β) →moving(β)×MC(β)	查询对象的将来运动
Temporal_selection	hf(α,β) ×interval→hf(α,β)	查询一个时间间隔内对象的运动，返回也是 HFDT 类型
Present	hf(α,β) →time	返回 HFDT 类型对象的目前时间
Life	hf(α,β) →interval	返回 HFDT 类型对象的整个生命周期(从运动开始到预测结束时刻)
Past_period	hf(α,β) →real	返回对象过去运动的时间长度
Future_period	hf(α,β) →real	返回对象将来运动阶段的时间长度
Past_projection	hf(α,β) →λ	返回对象过去运动的空间投影，结果为λ类型的空间对象
Future_projection	hf(α,β) →λ	返回对象将来运动的空间投影，结果为λ类型的空间对象

(注：α，β，λ是空间对象类型 points，lines，regions 中的一种)

通过表 3.2，可以知道使用 HFDT 数据类型描述时空对象的优点之一是：可以对时空对象在整个时间域内进行某些操作。当然，表 3.10 只是其中一小部分的操作，在以后的研究中可以扩充很多涉及到不确定性类型的操作，如 maybe、must_cross 等谓词。

3.5.2 支持运动中发生分形的数据类型

现实应用中，有些时空对象在某个时间段内可能表现为移动点，而在另一个时间段内转化

为移动线或者移动区域，如图 3.2 中复杂的时空对象，现有的数据类型就无法完整的表达这类运动过程中发生拓扑分析的时空对象。

图 3.3 运动中发生分形的时空对象

为了表示图 3.2 所示的时空对象，需要对数据模型进行扩充，提出新的时空类型来表达这类复杂的时空对象，基本思想是：将时空对象按某个时间段内表现的拓扑特征划分为若干子对象，每个子对象固定为一种拓扑形式的时空对象类型（基本的时空对象 mpoints，mlines 或者 mregions）。一个完整的时空对象需要用其所有的子对象来表示，很显然，这些子对象的时间域不会相交。给出了一类新的复合类型 CFMDT(composite fractal moving data type)来描述上述的时空对象。下面给出复合类型在离散层次上的定义。

定义 3.22 $D_{CFMDT(id,start,end)}$={(id,<(start1,end1,moving(α)),(start2,end2,moving(β))...
(startn,endn,moving(λ))>| n>0,($\forall i,j \in \{1,...,n\} : i<j \Rightarrow start_i < end_i < start_j < end_j$))}
其中$\forall \alpha,\beta,\lambda \in \{point s, lines, regions\}$

通过定义 3.22 可以看出，这类复合时空类型是有分布在不同时间区间内的若干个时空子对象组成的，例如将鸟类迁移的轨迹时而表现为移动线(所有的鸟排成一条线段飞行)，时而表现为移动区域(所有的鸟围成一个区域飞行)，这时就可以采用支持分形的复合数据类型来表示这类对象。有了复合时空数据类型，就需要对其提供各种操作算法。这里只是做一点理论的探讨，所以就不再继续论述。

3.6 时空数据操作的描述

3.6.1 投影操作

这类操作将移动对象作为操作数，根据属于时态数据类型的参数对象的时间属性或函数属性，计算不同类型的投影。

1. deftime： 返回移动对象有定义时间，结果为时间类型的区间集。
   deftime(α) → range(instant) α∈{mint,mreal,mbool,mpoints,mlines,mregions}
2. locations：返回 mpoints 对象投影后得到孤立点的集合，结果为空间类型 points。
   locations(mpoints) → points
3. traversed：返回 mregions 对象在平面上的投影，结果为空间类型 regions。
   traversed(mregions) → regions

4. trajectory：返回 mpoints 对象在二维平面上的轨迹，结果为空间类型 lines。

trajectory(mpoints) → lines

3.6.2 相交操作

1. atinstant： 返回移动对象在给定时刻的对象值，结果为 intime 类型。

atinstant (moving(α)×instant) → intime (α) α∈{int,real,bool,points,lines,regions}

2. atperiods：返回移动对象在给定时间间隔内的值，结果为移动对象类型。

atperiods (α×range(instant)) →α α∈{mint,mreal,mbool,mpoints,mlines,mregions}

3. initial，final：返回移动对象在定义时间内最早和最终时刻的值，结果为类型 intime。

initial (moving(α)) → intime (α) α∈{int,real,bool,points,lines,regions}

final (moving(α)) → intime (α) α∈{int,real,bool,points,lines,regions}

4. present：返回在某个时刻或时间间隔内移动对象是否有定义，结果为 bool 类型。

present (α×instant) → bool

present (α×range(instant)) → bool α∈{mint,mreal,mbool,mpoints,mlines,mregions}

3.6.3 谓词操作

1. disjoint：该类谓词检查两个参数对象在有定义的时间交集内是否满足空间拓扑关系 disjoint，即相离关系。

disjoint (α×β) →mbool α，β∈{mint,mreal,mbool,mpoints,mlines,mregions}

2. inside：该类谓词检查两个参数对象在有定义的时间交集内是否满足空间拓扑关系 inside，即包含关系。

inside (α×β) →mbool α，β∈{mint,mreal,mbool,mpoints,mlines,mregions}

3. intersects：该类谓词检查两个参数对象在有定义的时间交集内是否满足空间拓扑关系 intersects，即相交关系。

intersects (α×β) →mbool α，β∈{mint,mreal,mbool,mpoints,mlines,mregions}

4. meet：该类谓词检查两个参数对象在有定义的时间交集内是否满足空间拓扑关系 meet，即相切关系。

meet (α×β) →mbool α，β∈{mint,mreal,mbool,mpoints,mlines,mregions}

3.6.4 集合和聚集操作

1. intersection：该类操作计算两个参数对象在有定义的时间交集内的交集。返回的也是时空对象。

intersection (α×β) →λ α，β，λ∈{mint,mreal,mbool,mpoints,mlines,mregions}

2. union：该类操作计算两个参数对象在有定义的时间交集内的并集。返回也是时空对象。

union ($\alpha \times \beta$) $\rightarrow \lambda$ $\alpha, \beta, \lambda \in$ {mint,mreal,mbool,mpoints,mlines,mregions}

3. minus：该类操作计算两个参数对象在有定义的时间交集内的差集。返回也是时空对象。

minus ($\alpha \times \beta$) $\rightarrow \lambda$ $\alpha, \beta, \lambda \in$ {mint,mreal,mbool,mpoints,mlines,mregions}

4. min，max，avg：该类操作计算时态对象在定义时间内的最大、最小和平均值。

max (moving(α)) $\rightarrow \alpha$ $\alpha \in$ {mint,mreal }

min (moving(α)) $\rightarrow \alpha$ $\alpha \in$ {mint,mreal }

avg (moving(α)) $\rightarrow \alpha$ $\alpha \in$ {mint,mreal }

3.6.5 数值和距离操作

1. perimeter：返回 mregions 对象的周长，结果为时态类型 mreal。

perimeter (mregions) → mreal

2. area：返回 mregions 对象的面积，结果为时态类型 mreal。

area(mregions) → mreal

3. distance：该类操作计算两个参数对象在有定义的时间交集内对象间的最小距离。返回时态类型 mreal。。

distance (α) → mreal $\alpha \in$ { mpoints,mlines,mregions}

3.7 本章小结

本章给出了时空数据模型的抽象和离散模型描述。采用基于数据类型的时空数据建模的方法，首先定义基本数据类型和空间数据类型，通过引入类型构造子产生相应的时空数据类型，然后从抽象和离散的层次对数据类型进行了定义。结合时空数据库的实际应用需求，引入了两类复合时空数据类型，第一类可以方便的描述时空对象整个生命周期内的运动，既包括对象历史运动情况，也包括用数据类型的形式刻画将来的运动，而不是简单的通过函数预测的方式；第二类新的数据类型能表示更为复杂的、在运动过程中可能发生变形的时空对象。下一章将会根据本章提出的时空类型系统给出它们具体实现时的核心数据结构和典型操作的实现算法，完成了支持时空查询的数据类型和操作的定义，形成了时空查询语言。

第四章 时空数据库的查询语言

4.1 引言

时空数据库查询语言和数据模型密不可分，每种数据模型总是伴随着自己的查询语言。由于 SQL3 提供了抽象数据类型的扩展能力，只需要扩展数据类型和操作就可以形成新的支持时空查询的语言。在第三章中讨论了基于数据类型的时空数据模型和操作分类，本章将讨论时空数据类型的核心数据结构和典型操作的实现算法，这样就形成了支持时空查询的语言。

4.2 核心时空数据类型实现

本节主要介绍原型系统中时空数据类型的数据结构实现。时空对象的数据结构的实现采用了两种方式：基于 Vector[43]顺序存储和基于二级平衡二叉树 AVL 的存储。对空间对象的底层处理遵循 realm 的规范[16][17]。

4.2.1 空间数据类型

空间数据类型即空间 DBMS 中的空间数据类型，如 point，points，lines，regions 等。为了实现 lines 和 regions 类型，还需要定义 HalfSegment 类型，作为 lines 和 regions 的基本组成元素。

1．Point 类型对象的数据结构

```
typedef double Coord;
Class Point
{
Public:
    Point() {};
    Point( const bool d, const Coord& x = 0.0, const Coord& y = 0.0 );
    bool inside( Points& ps ) const; //判断该点是否落在点集对象 ps 内
    bool equal( const Point& p) const; //判断该点和 p 是否相等
    double distance( const Point& p ) const; //计算该点与点 p 之间的距离
    ...
Protected:
    Coord x;
    Coord y;
    bool defined; //是否定义
};
```

2．Points 类型对象的数据结构

```
Class Points{
Public:
    Points() {};
    Points( const int initsize );
    Points( Points& ps);
    const Rectangle<2> BoundingBox() const; //构建 Points 对象的最小闭包矩形 bbox
    bool Inside(Points& ps); //判断该点集是否落在点集对象 ps 内
    ...
    /*下面 6 个函数是对 points 对象的平衡二叉树结构进行遍历的外部接口*/
    Void SelectFirst(); //将当前指针指向 points 对象平衡二叉树结构的根部
    Void SelectNext(); //将当前指针指向 points 对象平衡二叉树结构的下一个 point 对象
    Void SelectLast(); //将当前指针指向 points 对象平衡二叉树结构的最后一个 point 对象
    Bool EndOfPt(); //判断遍历是否结束，即对象中是否有更多的 point 对象
    Void GetPt(Point &p); //获得 points 对象的平衡二叉树当前所指的 point 对象
    Void InsertPt(Point &p);   //将点对象 point 插入的 points 对象的平衡二叉树中
Protected:
    /*下面 6 个函数是对 points 类型平衡二叉树结构进行操作的内部接口*/
    int InsertAvl(Point& parent,Point& p,int root); //将点对象 p 插入 AVL-tree 结构中
    int GetHeight(Point& parent,int root); //获得 AVL-tree 树高
    void Rebalance(Point& parent,int& root); //对 AVL-tree 从新平衡
    int Balance(Point& parent,int root);
    int RemoveMin(Point& parent,int root,int& min); //删除 AVL-tree 中值最小的点对象
    int DeletFromPoints(Point& parent,int root,Point& p,int& del_idx); //删除指定的点对象

    Vector<Point> points;//组成点集 points 的点集合
    Rectangle<2> bbox;//points 对象的最小闭包矩形
    int pos;   //points 对象平衡二叉树结构中目前 point 对象的位置
    int rootidx; //points 对象平衡二叉树结构中根部 point 对象的位置
    int firstidx; // points 对象平衡二叉树结构中第一个 point 对象的位置
    int lastidx; // points 对象平衡二叉树结构中最后一个 point 对象的位置
};
```

3．HalfSegment 类型的数据结构

```
Class HalfSegment{
Public:
    HalfSegment() {}
    HalfSegment( const HalfSegment& hs );
    bool Intersects( const HalfSegment& hs) const;
    bool Overlap( const HalfSegment& hs) const;
    double Distance( const Point& p ) const;
    ...
Protected:
    bool defined; //是否定义
    bool ldp;
    Point lp;//左端点
    Point rp;//右端点
};
```

4．Lines 类型的数据结构

```
Class Lines{
Public:
    Lines (const int initsize);
    Lines (Lines& ls );
    const Rectangle<2> BoundingBox() const;
    bool Intersects(Lines & ls);
    float Length();
    ...
Protected:
    Vector<HalfSegment> lines; //组成线集的线段集合
    Rectangle<2> bbox;//lines 对象的最小闭包矩形
    int pos; //lines 对象平衡二叉树结构中指向目前半线段的位置
    int rootidx;
    int firstidx;
    int lastidx;
};
```

5．Regions 类型的数据结构

```
Class Regions{
Public：
    Regions(const int initsize);
    Regions(Regions& rs);
    const Rectangle<2> BoundingBox() const;
    void Get( const int i, HalfSegment& hs );
    void InsertHs( HalfSegment& hs );
    bool contain( const HalfSegment& hs );
    ...
Protected：
    Vector<HalfSegment> regions; //组成区域的线段集合
    Rectangle<2> bbox; //regions 对象的最小闭包矩形
    int pos; //regions 对象平衡二叉树结构中指向目前半线段的位置
    int rootidx；
    int firstidx；
    int lastidx；
};
```

Lines 和 Regions 类型数据结构中同样包含了 12 个与 points 结构类似的用于平衡二叉树存储结构处理操作的相关函数，这里没有详细列出。

4.2.2 区间集类型

1．区间类型的数据结构

为了避免重复编写代码，系统中大量采用了 C++中的模板 template。

```
template <class Alpha>
struct Interval{
    Interval() {};
    Interval( Interval<Alpha>& interval );
    Interval( Alpha& start,Alpha& end,const bool lc,const bool rc );
    bool operator==( Interval<Alpha>& i );
    bool Disjoint( Interval<Alpha>& i );
    void Intersection( Interval<Alpha>& i, Interval<Alpha>& result );
    ...
```

```
    Alpha start，end;
    bool lc，rc; //区间左右的开闭合性
};
```

2．区间集类型的数据结构

```
template <class Alpha> //Alpha 是组成区间集的基本类型
class Range {
Public：
    bool IsOrdered() const;
    void Add( Interval<Alpha>& i );
    void Get( const int i, Interval<Alpha>& ai );
    bool Before( Range<Alpha>& r );
    void Minus( Range<Alpha>& r, Range<Alpha>& result );
    ...
Protected：
    Vector< Interval<Alpha> > intervals;
    bool ordered;
};
```

经由 Range 类型可以定义时空数据库中使用的 RInt、RReal、Periods 等类型。

```
typedef Range<int> RInt;
typedef Range<real> RReal;
typedef Range<Instant> Periods;
```

3．intime(α)类型的数据结构

```
template <class Alpha> //Alpha 代表基本类型或者空间类型
struct Intime
{
    Intime( Instant& instant, Alpha& alpha ):instant( instant ),value(alpha),defined( true )
    ...
    Instant instant; //时间戳
    Alpha value; //值
    bool defined;
};
```

Intime(α)构造出的类型为二元组，表示的是一个时刻和对应的值的二元组。

4.2.3 单元类型

本小节的单元类型就是第三章中为了表示时空对象引入的快照类型的改进，因为单元类型可以通过函数的求值的方式给出两个快照时间戳之间任意时间点的类型值。

1．常值单元类型 ConstTemporalUnit 的数据结构

```
template <class Alpha>
struct ConstTemporalUnit
{
    virtual void TemporalFunction( Instant& t, Alpha& result ){
        assert(timeInterval.Contains( t ) );//判断 t 是否落在单元类型的时间定义域内
        result=constValue ;
    }
    Alpha constValue;
    Interval<Instant> timeInterval;
};
typedef ConstTemporalUnit<bool> UBool;
typedef ConstTemporalUnit<int> UInt;
```

UBool 和 UInt 表示 bool 和 int 类型构造出来的单元类型。通过 ConstTemporalUnit 的 TemporalFunction 函数可知，UBool 和 UInt 在时间戳之间的任意时间类型值是不发生变化的。

2．基于实数 Real 的单元类型 UReal

```
struct UReal
{
    virtual void TemporalFunction( Instant& t, Real& result ){
    double res = a * pow( t.ToDouble() - timeInterval.start.ToDouble(), 2 ) +
                    b * ( t.ToDouble() - timeInterval.start.ToDouble() ) +c;
    if( r ) res = sqrt( res );
    result.Set( true, res );
    }
    double a, b, c;
    bool r;
    Interval<Instant> timeInterval;
};
```

从 mreal 对象的单元类型 UReal 的 TemporalFunction 定义可以知道，以 a，b，c 为参数构

造二次多项式，在 ureal 内的任意时刻的值是该二次多项式的值或者是二次多项式平方根的值。是否采用平方根的值要根据实际应用情况。

3．基于空间点 point 的单元类型 UPoint

```
struct UPoint
{
    virtual void TemporalFunction( Instant& t, Point& result )
    {
        assert( t.IsDefined() );
        if( t == timeInterval.start ) result = p0;
        else if( t == timeInterval.end ) result = p1;
        else{
            Instant t0 = timeInterval.start;
            Instant t1 = timeInterval.end;
            double x = (p1.GetX() - p0.GetX()) * ((t - t0) / (t1 - t0)) + p0.GetX();
            double y = (p1.GetY() - p0.GetY()) * ((t - t0) / (t1 - t0)) + p0.GetY();
            result.Set( x, y );
        }
    }
    void Speed(UReal& result);
    void Distance( Point& p, UReal& result );
    Point p0, p1;
    Interval<Instant> timeInterval;
};
```

从空间点对象的单元类型 UPoints 的 TemporalFunction 定义可以知道，将 Upoints 的两个端点 p0 和 p1 连接成直线，根据该直线的斜率可以求得单元类型中的任意时间点的对象值。

由于单元类型 ulines 和 uregions 的结构和 upoints 类似，只是它们的 TemporalFunction 函数有所不同，这里就不再详述。

4.2.4 时空类型

1．时空和时态类型的快照数据结构

mint,mreal,mbool,mpoints,mlines,mregionis 等时空或时态类型是由其对象类型的快照集组成，而每种类型的快照集合又是建立在对应类型的单元类型之上。快照集合使用类 STL 中的 Vector 和 Set[43]两种数据结构存数单元类型。

```
template<class Unit,class Alpha>//Unit 代表单元类型，Alpha 代表对应的基本类型
class snapshot
{
public:
    void Get(const int i,Unit&upi);//取第 i 个单元
    void Add(unit& upi);//将单元类型对象 upi 加入单元存储结构
    int Position(const Instant&t);//确定 t 所在单元的索引号
    ……
private:
    Vector<Unit> v_unit; //轨迹单元 Vector 存储结构
    Set<Unit> s_units;//轨迹单元 Set 存储结构
    int curunit;//当前单元号
};
typedef Snapshot< UBool, bool > MBool;
typedef Snapshot< UReal, real > MReal;
typedef Snapshot< UInt, int > MInt;
```

2．移动点 mpoint 类型

```
class mpoint : public Snapshot< UPoint, Point >
{
public:
    mpoint() {}
    mpoint(int n): Snapshot < UPoint, Point >(n){} //构造一个含有 n 个快照的 mpoint 对象
    void Trajectory( Lines& lines ); //移动运动轨迹
    void Distance( Point& p, MReal& result );//移动点和点之间的距离
    void Distance( MPoint& mp,MReal& result );//移动点之间的距离
    void Speed(MReal& result); //移动点的速度
    void Inside(Regions& rs, Mbool& result); //判断移动点是否落在区域内
    void At(Instant t1,Points& ps); //返回时刻 t1 时移动点的位置
    ...
};
```

时态类型中的 mreal 以及时空类型中的 mlines、mregions 定义和 mpoint 相似，只是具体操作不同，这里不再累述。

3．HF 复合时空类型

```
template<class hm,class fm>//hm 代表过去运动类型，fm 代表将来运动类型
Class HF_obj
{
Public：
    Life(); //对象的整个生命周期
    Past();//返回对象过去运动的 hm 类型对象
    Future();//返回对象将来运动的 fm 类型对象
    ...
Protected：
    fm fm1;
    hm hm1;
};
```

4．CFMTD 时空复合类型

```
template <class Alpha> //子对象类型
Class sub_object
{
    //对子对象操作的函数
Protected:
    Instant start; //子对象开始时间
    Instatnt end;//子对象结束时间
    Alpha obj; //时空子对象的类型(mpoints，mlines，mregions)
};
```

CFMTD 时空复合类型数据结构

```
Class CFMTD
{
    //对 CFMTD 类型对象操作的函数
Protected:
    Instant start; //对象运动开始时间
    Instant end;//对象运动结束时间
    Vector<sub_object> cfmtd;
};
```

4.3 典型时空操作算法的实现

时空子系统主要实现了 5 类时空操作：(1)返回时空对象的操作，如求两个移动区域的交集，结果也为移动区域类型；(2)返回空间对象的操作，如求移动点的轨迹、瞬时值，返回值是分别为线对象和点对象；(3)返回单个数值的操作，如求两个点的距离，移动点的瞬时速度，结果为一个实数；(4)返回多个数值的操作，如求两个移动点间的距离，返回为 mreal 时态类型对象。

下面介绍一个典型的时空操作 mpr_inside，这是一个基于时间域的时空操作，用于判断一个 mpoints 对象在有定义的时间域内是否落在(inside)一个 regions 对象内。该类操作的实现思想是按照时间顺序遍历时空对象的时空快照，对快照和空间对象进行空间拓扑分析。算法伪代码描述算法 1 和算法 2。

算法 1 mpr_inside

```
Input: MPoints mp, Regions rs
Output: mbool
Begin
    mbool_init(&mb1); //初始化一个时态 bool 类型
    mSpatioFirst(mp);  //将当前指针 pos 指向对象快照存储结构的第一个快照
    mSpatioGetCurrentTime(mp,&start); //获取当前指针所指快照的时刻
    mSpatioLast(mp);  //将当前指针 pos 指向对象快照存储结构的最后一个快照
    mSpatioGetCurrentTime(mp,&end);
    while (start <= end){
        mSpatioGetValueByTime(mp, start,p); //获取时刻 start 的快照值，赋予对象 p
        bool inside = pr_inside(p,rs); //调用空间操作函数 见算法 2
        mbool_add(mb1,start,inside);
        mSpatioNext(mp);
        mSpatioGetCurrentTime(mp,&start);
    }
return mb1;
End.
```

算法 2 pr_inside

```
Input: Points p, Regions r
Output: p 是否完全落在 r 内部
Begin
Bool inside := TRUE;
```

```
SS_Reset();  /*初始化扫描线状态栈*/
PRSelectFirst(P, R, &object, &status);
While((status=end_of_none OR status=end_of_second) AND inside ){
 /* No point outside 'R' found and end of 'P' not reached. */
    if ( object = both OR object = second ){
        Halfsegment   h := Regions_GetHalfsegment(R);
        if ( h.left )SS_Add(h.s);    /*将对应线段的几何信息插入状态栈*/
        else SS_Del(h.s);    /*从状态栈中删除对应线段的几何信息*/
        else if ( object==first ){   /*扫描线与 p 接触，分析状态栈*/
            Segment h := SS_PredOfPoint(Points_GetPoint(P));
            if (h)inside := h.inside_Up;
            else inside := FALSE;
        }
        PRSelectNext(P, R, &object, &status);
    }
}
return inside;
End.
```

4.4 本章小结

本章首先介绍了时空数据模型中的主要数据类型对应的数据结构(存储结构)的实现，然后举例说明了经典的时空操作算法的实现思想，结合第三章定义的时空数据类型，数据类型和操作结合在一起扩充 SQL3 形成了支持时空查询的语言。下一章将集成之前的数据类型和操作到一个可扩充的数据库管理系统中，从而实现一个从底层提供支持时空类型和操作的 DBMS 原型系统 NHSTDB。

第五章 时空数据库管理系统 NHSTDB 设计与实现

通过第三、四章对基于数据类型的时空数据库数据模型、时空子系统内部核心数据结构、时空拓扑操作实现等问题的讨论，不但很好的解决了时空数据的表示问题，而且还有了高效、丰富的时空操作集来支持实用、复杂的时空查询。这些都为时空数据库管理系统的设计和实现奠定了底层基础。

实现时空数据库管理系统 NHSTDB 的具体技术路线是：在可扩充对象数据库 AMOSII 的基础上，通过自定义时空数据类型实现对时空对象的扩充，通过自定义外函数的手段来实现时空谓词，以支持时空操作。采用这种方案是因为：(1)课题组已开发出的基于数据类型的时空对象操作集已经比较完善，它不但解决了时空数据的表示问题，而且已经拥有高效的时空分析手段来支持复杂的时空对象查询；(2) AMOSII 是主存数据库，所以扩充后的原型系统 NHSTDB 也具备主存驻留特性，这样可以显著提高系统的查询效率。(3)AMOSII 对 SQL 的支持很完备，使得查询语言的实现很方便：它支持对数据类型和函数的扩充，能加快系统的开发进度。

5.1 AMOSII 简介

对象-关系数据库管理系统(Object-Relation DataBase Management System，简称 O-RDBMS)[23]是面向对象数据库系统(Object Oriented DBMS)和关系数据库系统的结合，既继承了关系数据库本身成熟的技术，又将面向对象方法引入到关系数据库中，增强了处理复杂对象等方面的能力。面向对象方法是一种支持模块化设计和软件重用的实际可行的编程方法，核心是充分支持用户定义类型、自定义函数、封装等特征。

AMOSII 是由瑞典 Uppasala 大学研制[44][45]的一种具有查询语言的对象-关系数据库系统，同时它也是一个主存 DBMS，相比基于硬盘的 DBMS 具有较高的存取速度。它采用了一种类似于 SQL 的结构化查询语言 AmosSQL 来定义、提取和操作数据。

5.1.1 AMOSII 的数据模型

AMOSII 的数据模型由三个基本元素组成的，分别是对象(object) 、类型(type) 和函数(function)。类型用于区分对象，对象则是类型的实例，对象的属性和对象间的关系都用函数来表示。

1. 对象

AMOSII 中含有两种类型的对象，文字对象(literal object)和代理对象(surrogate object)。文字对象比较简单，是表示字符串和数字的对象；所有的代理对象都被分配对象标识符(OID)以被系统管理。代理对象是复杂的，通常表示现实世界中的事物，如一座城市，一辆汽车，也可以

是一场足球比赛。系统内置的对象称为元对象(Meta object)，如类型(type)和函数(function)。AMOSII 中所有事物都被视为对象，可以将其类比为关心数据库中的元组。在 AMOSII 中创建一个对象的语法如下：

create <type name> instances: <object name>;

2．类型

存在一个对象，则存在至少一种类型。在 AMOSII 中 type 之间有 parent-type/child-type 的继承关系，也就是说如果一个 object 是一个 type 的一个实例，它也是这种 type 的所有 parent-types 的一个实例。创建 type 的 AmoSQL 语法如下：

create type <type-name>;

定义类型之间继承关系的语法如下：

create type <type-name> under <parent-type>;

3．函数

在 AMOSII 中，表示对象的属性、方法以及对象间的关系和对对象的查询，都是用函数来实现的。基本函数分为 5 类：存储函数(stored functions)，导出函数(derived functions)，外函数(foreign functions)以及代理函数(proxy functions)和数据库过程(database procedures)。

在系统 NHSTDB 中用到的主要是存储函数和外函数。存储函数通常用来标识 AMOSII 中对象的属性，比如 person 类型的对象，它一般具有姓名(name)和年龄(age)两个属性。在该对象上调用 name 存储函数就会得到 person 对象的名字。又比如空间 point 类型的一个对象，它具有 x，y 坐标这两个基本属性，调用 x 或 y 存储函数便可得到该 point 类型对象的 x 或 y 坐标值。

5.1.2 AMOSII 的 C 接口

AMOSII 系统给用户提供了两种不同类型的接口[46][47]，一种是应用程序接口 callin，通过 callin 接口应用程序访问 AMOSII 数据库；另一种类型的接口是系统级扩充接口 callout，通过 callout 接口 AMOSII 可以调用用 C 编写的外函数。

对 callin 接口来说，应用程序可以使用两种方式来和 AMOSII 通信。一种方法是利用嵌入式查询接口(embedded query interface)，将包含 AmoSQL 查询语句的字符串传递给 AMOSII，由 AMOSII 来分析判断，最后通过 AMOSII 给定的函数来获取结果，这种查询方式需要经过提交查询、词法分析、生成语法树直至查询，在效率上有一定的损失，但它可以支持用户提交很复杂的查询要求。另一种方法是快速通道接口(fast-path interface)，这种方法直接调用用 C 编写的函数，省去了在嵌入式查询接口方法中所要经过的步骤，效率比嵌入式查询接口方法要高很多。快速通道接口方法适合与 callout 接口配合使用，以实现系统级的扩充。

callout 接口是 AMOSII 扩充自身的一个方法，在 callout 中利用外函数可以达到扩充的目的。外函数区别于存储函数的一点就是：外函数实现对象的方法而存储函数实际上是实现存取对象

的属性。AMOSII 只支持存取属性这样的函数，对于复杂的函数它就交由用户去实现以实现对 AMOSII 的扩充。

5.2 NHSTDB 体系结构

5.2.1 系统结构

结合前人对时空数据库管理系统体系结构研究的优缺点，NHSTDB 系统采用扩展型体系结构，图 5.1 是时空数据库原型系统 NHSTDB 结构图。

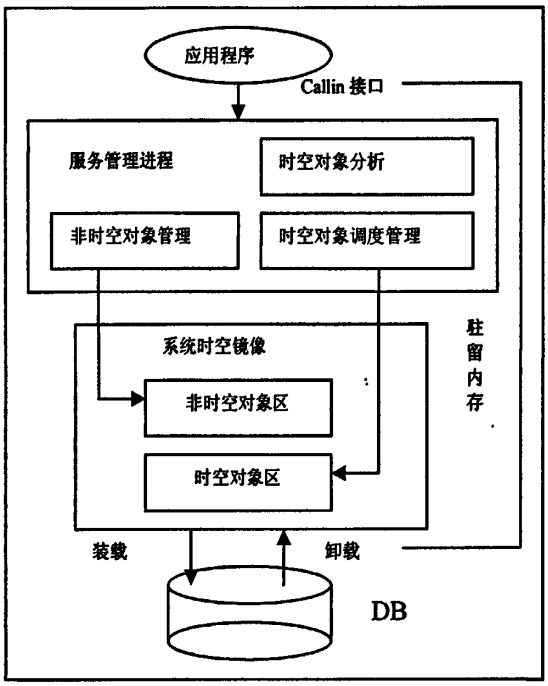

图 5.1 NHSTDB 结构图

如图 5.1 所示，在 NHSTDB 系统启动时，镜像文件(.dmp)会导入内存，与时空数据库服务管理程序共存于一个地址空间。这样，程序就可以方便的访问所需要的数据。在应用程序通过 callin 接口向数据库发起查询请求后，DBMS 经过词法分析、语法树生成、查询优化、查询计划生成阶段后，访问相应的内存数据。其中包括时空对象以及非时空信息。

对时空对象处理由“活动队列调度器”(5.2.3 节介绍)控制：从时空对象存储区获取必要的组成元素信息，在堆中建立对应的时空数据结构 STAVL（Spatio-Temporal AVL），将 STAVL 插入活动序列，运用 Realms 提供的时空分析函数操作 SAVL。随着时空查询请求的不断提出，活动序列将进行相应的。

如果内存镜像数据被修改，系统并不立即回写磁盘镜像文件，而是在系统退出的时候才进行回写，以提高效率。

5.2.2 时空对象的存储技术

AMOSII 的主镜像在运行时驻留内存。分析表明，在通常情况下，对象数据处于堆中(heap)。在对象数据存储区，每个对象拥有类似于结构体的内存布局。而为了实现时空对象的操作分析，则需要建立复杂的时空数据结构，NHSTDB 使用 STAVL 来表示时空对象。

要是在 DBMS 的对象数据存储区内直接存储上述的 STAVL，就必须设计复杂的对象结构，才有可能实现二级 AVL 的建立（因为对象存储区内存的布放由系统负责）。建立完毕的 STAVL 可以直接支持相应的时空分析操作。本方法可以概括为：复杂建模，无条件建树。

事实上，对于时空数据库的用户来说，只要知道某个时空对象的组成元素，并不需要知道它的具体底层实现细节。考虑到扩充时空数据库的目的是存储和操作空间数据，采用了更好的方法实现对象的建模和存储：在对象存储区中，只要保存时空对象的组成元素。在时空操作使用到对象时，再在内存的自由分配区建立 STAVL，两者通过指针联系。通过这个方案，对象存储区的对象布局就可以大大简化，而其对应的 STAVL 同样可以高效的访问。采用这种方案，对象的建模和存储都很简捷。

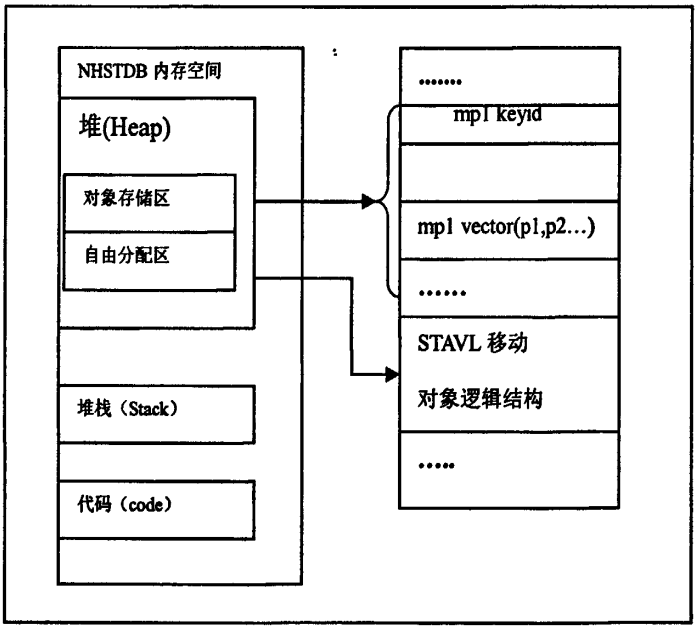

图 5.2 NHSTDB 内存布局图

NHSTDB 选用上述第二种存储技术，即："简单建模，必要时建树"，这种方法可以有效实现数据库建模与实现的分离。并且使系统具有很大的灵活性。图 5.2 是 NHSTDB 的内存布局图。由于每种时空数据类型在 DBMS 中只存储了其简要的时空数据信息，所以系统为每种类型提供了一个相应的建树算法，调用算法可以将时空信息在必要的时候转换为一个适合时空拓扑操作 STAVL 树。算法 3 和算法 4 给出系统 NHSTDB 中将移动点 mpoints 的时空信息建立在必要时建立树的算法。其它类型 mlines、mregions 和 mreals 等与之类似。

算法3： mpoints_make_tree_inline

```
Input：oidtype o // NHSTDB中mpoints对象的标示
Output：mSpatio *mps_new //指向mpoints对象的STAVL树的指针
Begin
    dcl_tuple(tvec);
    dcl_tuple(vvec); //声明存储时间戳和对应值的元组
    mSpatio *mpoints_new;
    tvec = mpoints_get_tsets_inline(o,&ifnull1);
    vvec = mpoints_get_vsets_inline(o,&ifnull2); //获取mpoints对象的tsets和vsets
    mpoints_new = mpoints_make_tree_in(tvec,vvec); //调用算法4
    return mpoints_new;
End.
```

算法4： mpoints_make_tree_in

```
Input：a_tuple tvec,a_tuple vvec // 存储时间戳和对应值的元组
Output：mSpatio *mps_new //指向mpoints对象的STAVL树的指针
Begin
    int tnum0 = a_getarity(tvec,FALSE); //获得tvec时间戳数量
    int vnum0 = a_getarity(vvec,FALSE); //获得vvec值数量
    mSpatio * mps_new = mSpatioOpen(); //在内存中初始化一个时空对象
    for (int i=0; i<tnum0;i++){ //遍历tvec和vvec元组
        a_getstringelem(tvec,i,time_string,100,FALSE);
        a_assign(v,a_getobjectelem(vvec, i, FALSE));
        Points *ps = points_make_tree_inline(v,&ifok); //调用points对象的建树函数
        mSpatioAdd(mps_new, t, (OID)ps); //将t时刻的点集对象ps插入到mps_new对象中
    }
    return mpoints_new;
End.
```

5.2.3 活动序列调度器

5.2.3.1 调度器原理

NHSTDB 采用的是层次时空对象结构，由基本时空元素组成系统时空对象，这些必要信息均存储在“对象存储区”。系统扩充的时空操作外函数在工作时，首先从“对象存储区”中寻找

对象的组成信息，在内存堆区(heap)的自由分配区建立与 Realms 中该对象类型一致数据结构的 STAVL，在时空数据结构 STAVL 建立完毕后，利用 Realms 提供的具体算法解决时空对象的拓扑分析。

根据局部性原理，目前经常使用的内存区，在可以预见的下一段时刻仍然很可能频繁使用。如要查询“在 T1 时刻在区域 R1 的所有移动点对象”。外函数为 R1 建立 SAVL 与 mpoint1 对象进行分析后，很可能要与 mpoint2、mpoint3 等多个移动点对象继续作分析。因此，为 R1 已经建立好的 STAVL 就没有必要立即释放，可以直接留用。

设立调度序列的目标是为了提高系统运行效率，需要考虑时间和空间因素的平衡。如果把所有时空对象的 STAVL 都置于序列中，可以省去以后建树过程，但是空间占有太多。如果每次使用完后都将建立的 STAVL 释放，则频繁建树的时间浪费很大。所以要根据实际情况设置序列的大小，不能过大也不能过小。

以下是活动序列的调度规则：

（1）当外函数需要操作某个时空对象，而序列里没有对应节点时，建立二级 STAVL，插入序列首部，保存指向 STAVL 的地址指针。如果序列已满，则删除队尾节点，释放对应 STAVL。

（2）外函数需要操作某个时空对象，而序列中有对应节点时，将其移至序列首部，并取出对应的指向 STAVL 的地址指针，交给 Realms 进行时空操作。

（3）当某时空对象的几何构造变化时，应在序列里删除对应节点，释放 STAVL。这样可以保证序列中的 STAVL 内容和“对象存储区”数据的一致性。

5.2.3.2 实验结果

实验室 1、测试 mpr_inside 操作(判断一个移动点对象 mp1 是否一直落在区域对象 rs1 内)在设置活动序列调度器和没有设置活动序列调度器的情况的效率。

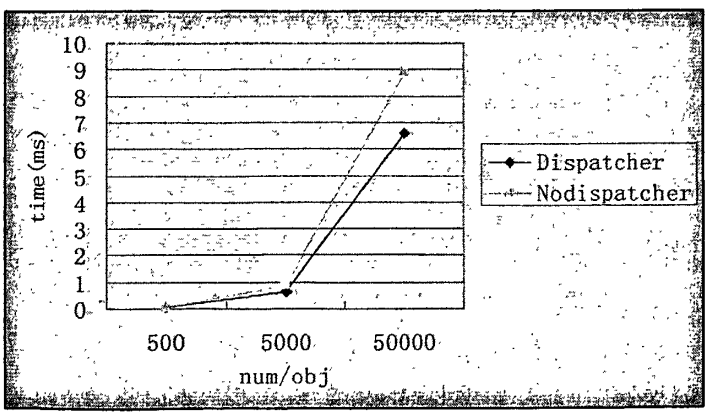

图 5.3 序列调度器对查询的效率影响

从图 5.3 可以看出，随着时空对象的组成变得更加复杂时，这个效率将显现的更加明显，因为当对象变的复杂时，用于建立 STAVL 树的代价将会变的很高。但是这种设计也存在一定的

问题，要确定活动序列调度器的大小，在下面一个实验中进行了了中进行确定。

实验二：测试 mpr_inside 操作在不同的活动序列调度器大小下的效率情况：(图 5.4，图 5.5，图 5.6 分别是对象点集合大小为 500、5000、50000 的情况设置不同序列调度器的时间效率情况)。直观上可以得出：序列调度器的大小设为对象点集个数的十分之一效果最好。

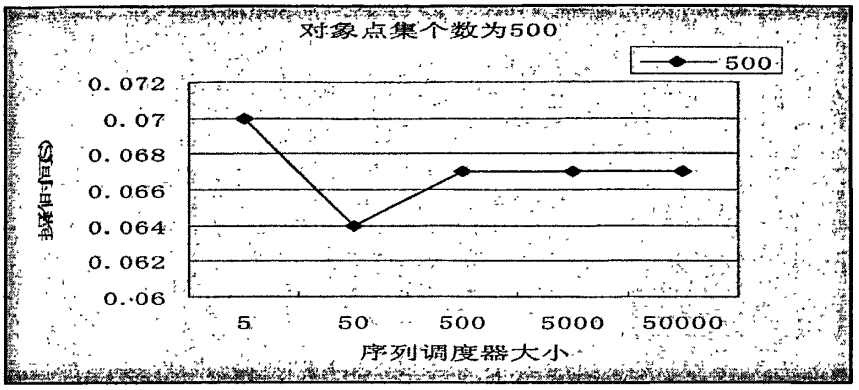

图 5.4 点集点数为 500 时操作效率

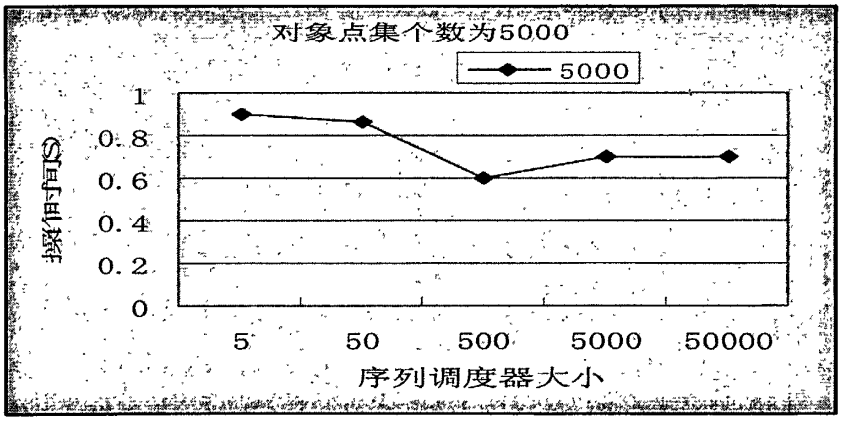

图 5.5 点集点数为 5000 时操作效率

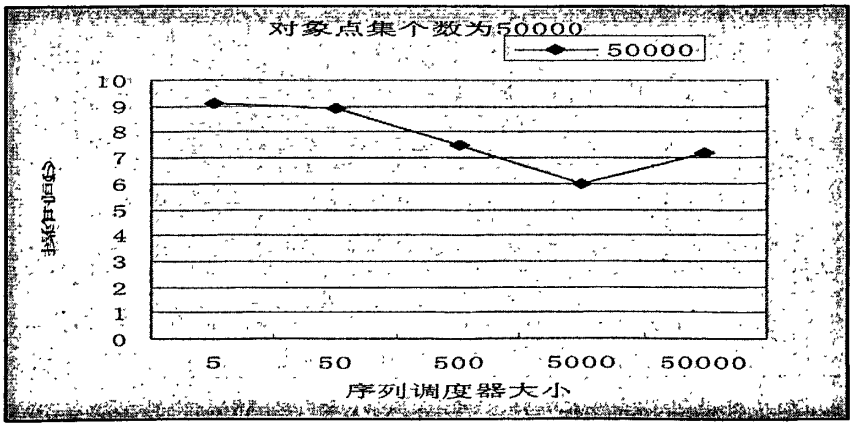

图 5.6 点集点数为 50000 时操作效率

5.3 NHSTDB 时空类型系统

NHSTDB 中包含了丰富的数据类型，除了基本数据类型如 int，real，boolean，string 外，还包含空间数据库所需要的 points、lines、regions 等空间类型，以及时空数据库所需要的 mpoints、mlines、mregions、mreal 等时空对象数据类型和时态数据类型 mreal。其中时空类型是在基本的空间元素和时间戳基础上形成。下图是时空类型的表示体系。关于 R-block、R-cycle、R-face 等概念可以参考[16][17]，这里不再详细累述，下面把重点放在对空间以及时空类型的表到上。

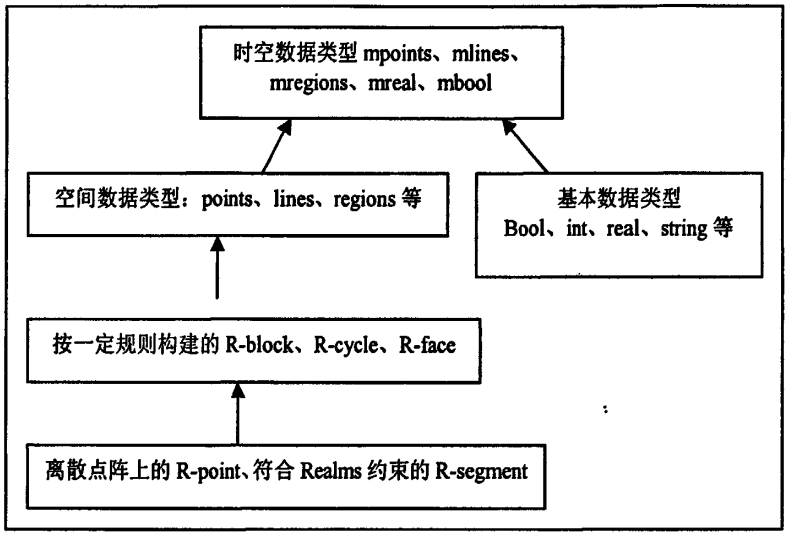

图 5.7 NHSTDB 时空类型的表示体系

5.3.1 基本空间元素 point 和 segment

Point 和 segment 对象分别表示空间点和空间线段，它们是其他空间类型的基础。表 5.1 和表 5.2 分别是 point 和 segment 类型的存储函数(基本属性)。

表 5.1 point 的存储函数

point 存储函数定义和类型	作用
Keyid：integer key	主键，唯一标示 point 对象
x：real	point 对象的 x 轴坐标
y：real	point 对象的 y 轴坐标

表 5.2 Segment 的存储函数

Segment 存储函数定义和类型	作用
Keyid：integer key	主键，唯一标示 segment 对象
Pfrom：point	Segment 对象的起点坐标
Pto：point	Segment 对象的终点坐标

5.3.2 空间类型体系设计

时空数据库管理系统NHSTDB提供了3个内置系统空间类型：表示点集合的points类型、表示线段集的lines类型以及表示区域集的regions类型。

在讨论空间类型的设计前，有必要给出AMOSII的内置数据类型Vector。Vector表示向量，也就是一个长度可变的数组。可以把Vector理解为数据结构中的容器，可以存储简单的文本对象，如数字，字符串，时间；也可在其中存放更为复杂的对象。本系统将空间基本元素point、segment存入Vector中，表示其属于某个空间对象。在NHSTDB中，空间类型(points、lines、regions)、时空类型(mpoints、mlines、mregions)、时态类型(mreal，mint)中均拥有属性（即存储函数）sets，sets的值类型就是这里说的Vector，可以存放组成空间类型的point对象（组成点集）和segment（组成线段集合区域集），对于时空类型就是存放空间对象以及其快照的时间戳。

考虑到空间类型points、lines和regions有很多相同之处：它们的基本存储函数都是keyid、sets；它们都需要有keyid_to、change_keyid、init等普通外函数（基本方法）。因此根据面向对象原理，需要首先设计一个基类型geo，再从geo派生出points、lines、regions。AMOSII支持派生，派生类型拥有基类的函数（包括存储函数和外函数），相当于面向对象中的“public继承”。

5.3.2.1 空间类型基类geo设计

Geo类型是points、lines、regions的基类，表5.3为 Geo的存储函数。

表5.3 Geo的存储函数

Geo存储函数定义和类型	作用
Keyid：integer key	主键，唯一标示geo对象
Sets：vector	标示geo对象的组成元素集合

Geo类型拥有很多外函数，可以通过这些函数处理对象的初始化、属性查询和修改、对Vector中元素的处理。表5.4为Geo的部分普通外函数。

表5.4 Geo的普通外函数

外函数定义	参数	返回值	作用
Keyidd_to	Integer	Geo	通过主键keyid，获取对象
change_keyid	Geo，integer	Geo	修改对象的keyid
delete_bykeyid	Integer	Integer	根据主键值，删除对象
set_sets	Geo，Vector	Geo	设置对象的组成元素集合
add_sets	Geo，vector	Geo	添加新的对象组成元素
del_sets	Geo，vector	Geo	减少对象的组成元素集合
clear_sets	Geo	Geo	清空对象的组成元素集合
init	Integer，Vector	Geo	初始化对象并赋值

5.3.2.2 空间类型 points 设计

points 表示几何上的点集，如一个机群。其拥有的 Vector 中元素都是 point 对象。它继承自 geo，拥有基类型 geo 的所有存储函数和外函数。按照 AmosQL 语法对 Points 定义为：Create type points under geo；

除了继承得来的属性和方法外，NHSTDB 还为 points 类型实现了 8 个常用的空间函数，用于分析对象间的拓扑关系或者获得空间对象的一些属性值如点集中点的数量。

表 5.5 points 空间分析函数

外函数定义	参数	返回值	作用
pp_equal	points，points	0/1	判断点集 ps1 是否等于点集 ps2
point_in_points	point，points	0/1	判断点 p1 是否属于点集 ps1
pp_disjoint	points,points	0/1	判断点集 ps1 是否相离于点集 ps2
p_count	points	Integer	返回点集的组成点数量
p_diameter	points	real	返回点集中最远两点间距离
pp_intersection	points，points	points	求两个点集的交，返回结果集的对象 oidtype
pp_plus	points，points	points	求两个点集的并，返回结果集的对象 oidtype
pp_minus	points，points	points	求两个点集的差，返回结果集的对象 oidtype

5.3.2.3 空间类型 lines 设计

在系统 NHSTDB 中，lines 类型用来描述空间线段集，比如一条河流及其分支。一个 lines 对象由多条线段（segment 对象）组成。Lines 的定义为：Create type lines under geo；

系统中为 lines 类型实现了 14 个空间分析外函数，表 5.6 只列出其中常用的几个。

表 5.6 lines 空间分析函数

外函数定义	参数	返回值	作用
l_length	lines	real	返回线集 ls1 的组成线段长度之和。
l_diameter	lines	real	返回线集 ls1 中最远两端点的距离。
l_count	lines	real	返回线集 ls1 的组成线段数量。
ll_meets	lines,lines	0/1	判断线集 ls1 是否与线集 ls2 相切
ll_intersects	lines，lines	0/1	判断线集 ls1 是否与线集 ls2 相交
ll_border	lines,lines	0/1	判断线集 ls1， ls2 是否有共同线段
ll_intersection	lines,lines	points	求两个线集 ls1，ls2 的交点集合。返回结果点集的 oidtype
ll_minus	lines，lines	lines	求两个线集 ls1，ls2 的差集。返回结果线集对象的 oidtype
ll_plus	lines，lines	lines	求两个线集 ls1，ls2 的并集。返回结果线集对象的 oidtype

5.3.2.4 空间类型 regions 设计

在系统 NHSTDB 中，regions 类型用来描述封闭空间，比如一个国家的疆域。一个 regions 对象由多条线段（Segment 对象）组成，但是要求这些线段在几何上封闭（允许用户输入不封闭的线段组，但这样就起不到原有的作用）。按照 AmosQL 语法对 Regions 的定义：Create type regions under geo；

regions 拥有 28 个空间分析外函数，表 5.7 列出部分空间外函数。

表 5.7 regions 空间分析函数

拓扑分析函数	参数	返回值	作用
pr_inside	points，regions	0/1	判断点集 ps1 是否落在区域集 rs1 内
rr_meets	regions，regions	0/1	判断区域集 rs1，rs2 是否相切
lr_intersects	lines,regions	0/1	判断线集 ls1 和区域集 rs2 是否相交
rr_intersection	regions，regions	regions	求两个区域集 rs1，rs2 的交。返回结果区域集的 oidtype
r_area	regions	real	返回区域集 rs1 的面积
rr_plus	regions，regions	regions	求两个区域集 rs1，rs2 的并。返回结果区域集的 oidtype
rr_minus	regions,regions	regions	求两个区域集 rs1，rs2 的差。返回结果区域集的 oidtype
r_count	regions	Integer	返回区域集 rs1 的组成线段数量
r_diameter	regions	real	返回区域集 rs1 中最远两个端点距离

5.3.3 时空类型体系设计

时空类型是建立在空间数据类型和普通数据类型的基础之上的。每个时空类型与其对应的空间类型之间通过时空类型的 vsets 属性相连接。空间类型负责底层的空间属性数据的存储，而每个时空类型则表示了由这些底层数据按照时间点顺序组成的时空对象集。

5.3.3.1 时空类型 mpoints 设计

mpoints 即时空点集是由若干不同时刻的点集对象组成的，例如飞行中的机群、行走中的人群。按照 AmoSQL 语法对 mpoints 的定义如下：create type mpoints。

mpoints 有三个基本属性（存储函数），通过这三个属性的值在进行拓扑分析时使用函数 mpoints_make_tree_inline(oidtype obj)生成 STAVL 树。表 5.8 是 mpoints 的存储函数。

表 5.8 mpoints 的存储函数

Mpoints 存储函数定义和类型	作用
Keyid：integer key	主键，唯一标示一个 mpoints 对象
tSets：vector	标示 mpoints 对象采样点的时间值的集合
vSets：vector	标示 mpoints 对象采样点的 Points 对象的集合

mpoints 类型提供了数十个外函数，可以处理 mpoints 对象的初始化，属性的查询、修改等普通应用，也可以实现对 mpoints 对象进行时空操作。表 5.9 给出了几个常用的时空操作函数。

表 5.9 mpoints 的外函数

Mpoints 外函数(方法)定义	参数	返回值	作用
mpp_distance	mpoints，Points	mreal	求移动点 mps1 和点对象 ps2 之间距离，返回 mreal 类型的对象。
mpmp_distance	mpoints，mpoints	mreal	求移动点 mps1 和移动点 mps2 之间距离，返回 mreal 类型的对象。
mpl_distance	mpoints，lines	mreal	求移动点 mps1 和线对象 ls2 之间距离，返回 mreal 类型的对象
mpr_distance	mpoints，regions	mreal	求移动点 mps1 和区域对象 rs1 之间距离，返回 mreal 类型的对象
mpl_on	mpoints，lines	mbool	判断移动点 mps1 是否在线对象 ls1 上。返回 mbool 类型对象。
mpp_disjoint	mpoints，points	mbool	判断移动点 mps1 是否和点集对象 ps1 相离。返回 mbool 类型对象。
mpmp_disjoint	mpoints，mpoints	mbool	判断移动点 mps1 是否和移动点 mps2 相离。返回 mbool 类型对象。
mpp_equal	mpoints，points	mbool	判断移动点 mps1 是否和点集对象 ps1 相等。返回 mbool 类型对象。
mpmp_equal	mpoints，mpoints	mbool	判断移动点 mps1 是否和移动点 mps2 相等。返回 mbool 类型对象
mpr_inside	mpoints，regions	mbool	判断移动点 mps1 是否落在区域对象 rs1 内。返回 mbool 类型对象
trajectory	mpoints	lines	求移动点 mps1 的运动轨迹。返回为 lines 对象。
duration	mpoints	real	求移动点有定义的时间域，返回为 real 类型值
mp_start	mpoints	real	求移动点有定义的起始时间，返回 real 类型值
mp_stop	mpoints	real	求移动点有定义的结束时间，返回 real 类型值
mp_at	mpoints，time1	points	求移动点 mps1 在某个时间点时位置。返回值为 points 对象。

5.3.3.2 时空类型 mlines 设计

mlines 即时空线段集类型是由若干不同时刻的线段集对象组成的，例如飞机所飞行的路线，在不同时刻飞机处于不同的路线上。按照 AmoSQL 语法对 mlines 的定义：create type mlines。

mlines 和 mpoints 类似，具有三个基本属性（存储函数），通过这三个属性的值可以在进行拓扑分析时使用函数 mlines_make_tree_inline(oidtype obj)将 mlines 对象生成一棵 STAVL 树。表 5.10 就是 mlines 的存储函数。

表 5.10 mlines 的存储函数

Mlines 存储函数定义和类型	作用
Keyid：integer key	主键，唯一标示 mlines 对象
tSets：vector	标示 mlines 对象采样点的时间值的集合
vSets：vector	标示 mlines 对象采样点的 lines 对象的集合

表 5.11 是 mlines 类型提供的时空操作外函数。

表 5.11 mlines 的外函数

mlines 外函数(方法)定义	参数	返回值	作用
mpml_distance	mpoints，mlines	mreal	求移动线 mls1、ls2 的距离，返回 mreal 类型
mlp_on	mlines，points	mbool	判断点集对象 ps1 是否在移动线集 mls1 上
mll_disjoint	mlines，lines	mbool	判断移动线集 mls1 是否和线集 ls2 相离
mlml_disjoint	mlines，mlines	mbool	判断移动线集 mls1 和移动线集 mls2 是否相离
mll_meets	mlines，lines	mbool	判断移动线集 mls1 是否和线集 ls2 相切
mlml_meets	mlines，mlines	mbool	判断移动线集 mls1 和移动线集 mls2 是否相切
mll_intersects	mlines，lines	mbool	判断移动线集 mls1 是否和线集 ls2 相交
mlml_intersects	mlines，mlines	mbool	求移动线集 mls1 和移动线集 mls2 是否相交
mll_equal	mlines，lines	mbool	求移动线集 mls1 是否和线集 ls2 相等
mlml_equal	mlines，mlines	mbool	求移动线集 mls1 和移动线集 mls2 是否相等
mlr_inside	mlines，regions	mbool	求移动线集 mls1 是否落在区域集 rs2 内
mlr_meets	mlines，regions	mbool	求移动线集 mls1 是否和区域集 rs2 相切
mlr_intersects	mlines，regions	mbool	求移动线集 mls1 是否和区域集 rs2 相交
mpml_on	mpoints，mlines	mbool	求移动点集 mps1 是否在移动线集 mls2 上

5.3.3.3 时空类型 mregions 设计

mregions 即时空区域集是由若干不同时刻的区域集对象组成的，例如台风的范围，在不同

时刻台风处于不同的范围。按照 AmoSQL 语法对 mregions 的定义：create type mregions。mregions和mlines、mpoints类似，具有三个基本属性（存储函数），通过这三个属性的值可以在进行拓扑分析时通过使用函数mregions_make_tree_inline(oidtype obj)将mregions对象生成一棵STAVL树。表5.12就是mregions的存储函数。

表 5.12 mregions 的存储函数

mregions 存储函数定义和类型	作用
Keyid：integer key	主键，唯一标示 mregions 对象
tSets：vector	标示 mregions 对象采样点的时间值的集合
vSets：vector	标示 mregions 对象采样点的 regions 对象的集合

表 5.13 是 mregions 类型提供的时空操作外函数。

表 5.13 mregions 的外函数

mregions 外函数(方法)定义	参数	返回值	作用
mpmr_distance	mpoints，mregions	mreal	求移动点 mps1 和移动区域 mrs2 的距离，返回 mreal 类型的对象
mlmr_inside	mlines，mregions	mbool	求移动线 mls1 是否在移动区域 mrs2 内
mlmr_meets	mlines，mregions	mbool	求移动线 mls1 是否和移动区域 mrs2 相切
mlmr_intersects	mlines，mregions	mbool	求移动线 mls1 是否和移动区域 mrs2 相交
mpmr_inside	mpoints，mregions	mbool	求移动线 mls1 是否落在移动区域 mrs2 内部
mrr_equal	mregions，regions	mbool	求移动区域 mrs1 是否和区域集 rs1 相等
mrr_disjoint	mregions，regions	mbool	求移动区域 mrs1 是和区域 rs2 相离
mrr_inside	mregions，regions	mbool	求移动区域 mrs1 是否在区域 rs2 内
mrr_intersects	mregions，regions	mbool	求移动区域 mrs1 是否移动区域 rs2 相交
mrr_meets	mregions，regions	mbool	求移动区域 mrs1 是否移动区域 rs2 相切
pmr_inside	points，mregions	mbool	求点集 ps1 是否落在移动区域 mrs2 内
lmr_inside	lines，mregions	mbool	求线集 ls1 是否落在移动区域 mrs2 内
lmr_meets	lines，mregions	mbool	求线集 ls1 是否和移动区域 mrs2 相切
lmr_intersects	lines，mregions	mbool	求线集 ls1 是否和移动区域 mrs2 相交
mrr_intersection	mregions，regions	mregions	求移动区域 mrs1 和区域 rs2 的交集
mrmr_intersection	mregion，mregions	mregions	求移动区域 mrs1 和移动区域 mrs2 的交集
traversed	mregions	regions	返回 mregions 对象在平面上的投影，结果为空间类型 regions

5.4 NHSTDB 查询语言

由于时空数据库系统 NHSTDB 是用 AMOSII 做为研究平台的，而 AMOSII 提供了标准的面向对象 SQL。所以 NHSTDB 使用的查询语言基本上就是参照 AMOSQL 的规范定义的。NHSTDB 主要体现在提供了丰富的时空数据类型和谓词操作支持时空对象操作。这些类型和谓词已经在 5.3 节中详细描述过。这里给出常用的语句的 BNF 定义。

(1) 定义类型语句

create-type-stmt ：= 'create type' type-name ['under' type-name-list] ['properties' '(' atrr-function-list')'];

type-spec ：= type-name | 'Bag of' type-spec | 'Vector of' type-spec

attr-function ：= generic-function-name type-spec ['key']

如：create type Kid under Person properties (name Charstring key, attitude Integer);

(2) 删除类型语句

delete-type-stmt ：= 'delete type' type-name;

如：delete type Person;

(3) 创建对象语句

create_object_stmt ：= 'create' type-name ['(' generic-function-name-list')'] 'instances' initializer-list;

initializer ：= variable|[variable]'('expr-list')'

如：create Person(name,age) instance :duan('duan hailiang',26),:gao('gao liang',24);

(4) 删除对象语句

delete_object_stmt ：= 'delete' variable;

如：delete :duan;

(5) 查询语句

query_stmt ：= select_stmt | function_call | expr;

select_stmt：='select' ['distinct'] expr_list [into_clause] [from_clause] [where_clause];

into_clause ：= 'into' variable_list;

from_clause ：= 'from' variable_declaration_list

where_cluse ：='where' predicate_expression

variable_delaration ：=type_spec local_variable

predicate-expression ：= predicate-expression 'and' | 'or ' predicate-expression | simple-predicate

simple-predicate ：: = expr | expr comparison expr

comparison ：=< | >|<=|>=|!=

如：select distinct friends(p) from Person p where name(parents(p))='john';

(6)更新语句

update-stmt ：= update_op update-item [from-clause] [where-clause]

update-op ：='set' | 'add' | 'remove'

update-item ：= function-name '(' expr-commalist ')' '=' expr

如： set name(:duan) = 'duan haijun';

add friends(:duan) = 'linux';

remove friends(:duan) = 'linux';

5.5 系统使用示例

在前面介绍过的时空类型的基础上，本节通过示例具体讨论如何利用 NHSTDB 为具体的时空应用服务。

应用环境设定如下：飞机场 airportA，airportB，每个飞机场有若干个航班 fights 起飞，另有天气状态类型如高压区、雷暴区域、龙卷风区域、等温区域等。

这样一个设定情形是日常生活中经常遇到的，而且也需要设计一个系统来实时的查询相关的信息，如获得雷暴区域影响的飞机场、距离最近的飞机场等。这些查询可能只涉及空间对象的信息，更有可能涉及时空对象的查询。

5.5.1 示例数据库逻辑设计

在关系型数据库逻辑设计阶段，要设计用户表和属性。作为对象关系数据库，NHSTDB 的逻辑设计主要是设计用户类型和相应的存储函数（属性)。系统拥有系统空间类型 points、lines 和 regions 以及时空类型 mpoints、mlines 和 mregions，当然还提供了一些其他的辅助类型如基本空间元素类型 point、segment 和 mreal 等类型。用户可以将其视为和 number、real、charstring 等一样的内置类型。建立如下用户类型：airport、weather、fights（具体定义语句省略)。

表 5.14 示例数据逻辑设计

类型名	存储函数名（属性）	取值类型
Airport	Name	Charstring
	Location	points
Flights	Id	Charstring
	From	Charstring
	To	Charstring
	Route	mpoints
Weather	Kind	Charstring
	Area	mregions

类型 Airport 用来记录特定的地区位置，属性 Name 表示机场名，属性 Location 用来表示飞

机场位置信息，是一个空间类型。在类型 flight 中除了用于表示飞机的 id 和 from 等基本属性外，属性 route 是一个移动点类型，表示该飞机的飞行过程。在类型 weather 中，kind 属性用来表示天气类型，如雷暴、雨雪等，而 area 是一个移动区域类型，用来表示天气范围的变化。

以 flights 为例，给出用户创建时空对象的步骤：

● 定义 flights 类型和存储函数：

```
Create type flights;
Create function id(flights)->charstring as stored;
Create function from(flights)->charstring as stored;
Create function to(flights)->charstring as stored;
Create function route(flights)->mpoints as stored;
```

● 建立 flights 对象 f1：

```
Create fights instances :f1;
```

●给对象 f1 赋值（包括时空属性和非时空属性）：

给这个对象的 id 属性赋值：set id(:f1)='SH1001';

给这个对象的 from 属性赋值：set from(:f1)='ShangHai';

给这个对象的 to 属性赋值：set to(:f1)='BeiJing';

给这个对象的 route 属性赋值（即存储 f1 的几何信息）：

```
Init_point(1,'p1',118,32); /*创建 1 号 Point 对象*/
Init_point(2,'p2',202.40); /*创建 2 号 Point 对象*/
...
Init_point(100,'p100',1044,5034); /*创建 100 号 Point 对象*/
mpoints_init(1,vector(point_keyid_to(1),point_keyid_to(2),... ,point_keyid_to(100)),
                vector('10:00','10:02',... '11:30')); /*创建 1 号的移动点对象*/
set route(:f1)= mpoints_keyid_to(1); /*给航班 f1 的 route 属性赋值*/
```

在以上的命令中，init_point、point_keyid_to、mpoints_keyid_to、mpoints_init 等均为系统 NHSTDB 时空类型的外函数，用户在系统命令中可以直接使用。建立其他时空对象的方法与上面相同。在所有时空对象创建完毕之后，原型系统中就记录下了上述设计的应用环境。

这里需要强调的一点是，在 realm 规范下，为了保证拓扑分析的健壮性和正确性，对空间对象有严格的要求，详细见参考文献[16][17]，但是这里假设用户并不了解这些约束。而对对象之间的调整操作由原型系统本身在系统内部实施，用户不需要了解 Realms 底层细节。

5.5.2 示例数据库查询

下面给出原型系统的具体查询示例。从以下具体查询语句可以发现，在原型系统中，时空

类型和普通类型的操作方式完全一样。灵活使用时空类型的操作函数，可以有效支持复杂的时空应用。查询语句中加黑的单词是 NHSTDB 提供的时空查询谓词。

例 1：查询所有从 NanJing 出发的航线超过 5000km 的航班？

```
SELECT id(f1),l_length(trajectory(route(f1)))
FROM flights f1
WHERE from(f1)=' NanJing'  AND l_length(trajectory(route(f1))) > 5000;
```

例 2：查询所有从 NanJing 出发的飞机的出发时间和到达时间？

```
SELECT id(f1),mp_start(route(f1)), mp_stop(route(f1))
FROM flights f1
WHERE from(f1)='NanJing';
```

例 3：查询所有从 NanJing 出发航程小于 2 小时的航班？

```
SELECT from(f1),to(f1),duration(route(f1)
FROM flights f1
WHERE from(f1)='NanJing' AND duration(route(f1)) <= 2.0;
```

例 4：找出所有两架飞机航线间的距离小于 500m 的航班对？

```
SELECT id(f1)，id(f2)
FROM fights f1，flights f2
WHERE id (f1) <> id(f2) AND minvalue(mpmp_distance(route(f1),route(f2))) < 500;
```

这里 mpmp_distance 是求两个移动点间的距离，minvalue 求一组值中的最小值，这是一个典型的移动对象间对象连接操作。

例 5：查询通过雪暴天气区域的航班？

```
SELECT id(f1)
FROM flights f1，weather w1
WHERE kind(w1)='snow storm'  AND duration(visits(route(f1),area(w1)))>0;
```

这里 visits 操作求移动点 route 穿过移动区域 area 的情况，返回的是一个移动点。Duration 求移动点的生命周期。假如一个航班穿过了一个雷暴区域，则 visits 返回的移动点为非空对象，也就是说返回的移动点的生命周期大于零。

例 6：查询所有被雪暴天气影响的飞机场

```
SELECT DISTINCT Name(a1)
FROM airport a1，weather w1
WHERE kind(w1)='snow storm'  AND
        pr_inside(location(a1),traversed(area(w1))) = TRUE;
```

这里 traversed 计算移动区域移动过的区域之和，pr_inside 操作用来判断点对象是否落在区域对象内部。

例 7：查询从 Nanjing 机场出发的航班的总里程。

```
Sum(SELECT l_length(trajectory(route(fl)))
FROM flights fl
WHERE from(fl)='NanJing');
```

5.6 本章小结

本章首先介绍了主存数据库 AMOSII 的特性和功能。AMOSII 具有面向对象的特性，因此很适合存储空间以及时空对象，而它的可扩充特性则为原型系统时空谓词的实现提供了良好的平台。

NHSTDB 时空类型体系包括基本空间元素类型 point、segment；系统空间基类 Geo，系统空间类型 Points、Lines、Regions；系统时空类型 mpoints、mlines、mregions。其中 Points 的几何描述由 point 向量表示，Lines 和 Regions 几何描述由 segment 向量表示，mpoints 的几何描述由<points,instant>向量表示，mlines 的几何描述由<lines,instant>向量表示，mregions 的几何描述<regions,instant>向量表示。在这种类型结构下，用户只需要给出空间和时空对象的基本组成元素就能够有效描述其几何形态。然后由系统将其转化为第四章中给出的复杂、高效的时空类型数据结构，参与时空操作。

本章的最后给出了一个时空数据库的应用实例，从数据库查询实例可以看出，原型系统 NHSTDB 中的空间类型以及时空类型与普通类型的操作方式完全一样。灵活使用系统提供的时空对象分析外函数，可以有效支持复杂的时空应用。

第六章 总结和展望

6.1 论文总结

近些年，随着存储、传感器和数据库技术的发展，越来越多的时空信息被采集并需要进行处理，因此迫切需要高效的系统软件支撑技术。目前，对于这些时空信息的管理研究主要从应用程序角度考虑，针对不同的服务需要添加相应的应用程序，没有从系统软件角度出发设计和研究直接解决时空信息的管理和查询等各种问题，缺乏一个时空信息管理领域的数据库原型系统。应用程序无法对底层的数据进行管理和处理，并不能从根本上解决问题，在效率上也很难满足实际需要。同时，现有的各种数据库系统均不针对时空信息管理领域，在数据存储和查询处理效率上都有很多欠缺和不足。本文工作主要在于为时空信息提供建模技术和软件平台，解决若干时空数据库领域的核心问题。

本文主要做了三个方面的工作：

首先研究了基于数据类型的时空数据模型，扩充了现有的时空数据类型系统，提出了两类新的时空数据类型，它们能支持更加实用和复杂的时空对象。

其次，研究了时空数据类型的实现方法，给出了类型系统中核心的数据结构，在此基础上讨论了时空操作的分类并给出了典型的时空操作算法实现。

最后研究了对象关系 DBMS AMOSII，基于这个系统扩充实现了 NHSTDB 时空数据库管理系统，它能支持丰富的时空类型和时空操作。通过一个时空信息处理的应用实例展示了 NHSTDB 的易用性和高效性。

6.2 进一步研究与展望

本文虽然对时空数据库的一些关键问题做了有益的讨论和研究，但是还有很多关键问题需要进一步的研究。下面主要从三个角度对进一步的研究进行分析。

(1) 对复合时空数据类型进一步研究，给出复合类型详细的操作实现和性能比较。虽然本文对复合时空类型做了有益的讨论，也给出了其数据结构和相关操作，但是这些类型和操作还没有集成到 DBMS 中，需要对其进行继续的研究，给出完善的操作和高效的实现，进一步设计算法，开发出一个独立、完整的时空代数软件包 STAL（Spatio-Temporal Algebra software package）。

(2) 并行技术[48][49]能充分利用目前计算机机群高性能的 CPU、大容量内存、高速磁盘阵列以及高带宽通信网络所构成的高性能处理环境。在之前的研究中，课题组已经将并行技术在空间拓扑分析操作上应用做了深入的研究[50]。时空数据库就是要实现对大规模的时空信息进行处理的，为了提高时空对象的处理速度，将并行技术引入其中将是一个不错的尝试。

(3) 研究可视化时空数据库查询语言。可视化语言[51][52]可为查询谓词提供了方便和直观的图形化描述，并且支持用这些谓词形成时空查询。如果能在时空数据库中成熟的应用可视化语言，那将能大大的提高时空数据库的普及和应用前景。

6.3 对时空数据库的一点思考

全文到这里就要结束了，时空数据库管理系统 NHSTDB 是我研究生两年多来汗水和努力的结晶。从当初对实验室师兄文韬和许建秋的敬仰，羡慕他们能编写复杂的系统软件，尤其是空间数据库管理系统 NHSpatial，到自己阅读大量的论文、编写程序；从在老师的指导下对 AMOSII 的了解到实际上自己动手扩充数据库。两年来的一幕幕都感觉尽在眼前。清楚的记得 09 年暑期写完程序，通过测试用例的喜悦。唯一的遗憾是 AMOSII 内核并非开源，至今没有找到合适的方式进行索引的扩充。希望以后能找到理想的方式实现时空对象索引。

目前，从技术角度看我国的信息产业处于整个产业链的下游，因此在中上游领域中拥有自主创新的产品是我国信息化产业的重中之重。而时空数据库技术是在基础软件领域中处于中上游的核心技术，具有广泛的应用前景，对我国的信息产业的发展具有非常重要的意义。随着越来越多的采集时空数据信息的设备的普及，时空相关应用正在深入到我们身边，这些必将进一步促进时空数据库技术的不断发展。我们有理由相信，在不远的将来，当人们在享受丰富多彩的时空数据服务的背后，正是一批坚如磐石的高性能时空数据库系统撑起了时空应用服务的广阔天空。

所谓“不积跬步，无以至千里；不积小流，无以成江海”，我们的时空数据库原型系统 NHSTDB 虽然有待完善，但毕竟已经迈出了可喜的步伐。而随着交通网络、高性能并行技术的运用，我相信，更加完善的、能在实际应用中使用的时空数据库支撑平台诞生的日子已经不远了。

参考文献

[1] Silberschatz A，Zdonik SB. Strategic directions in database systems—Breaking out of the box. ACM Computing Surveys，1996，28(4)：764~778.

[2] A.U.Tansel, J.Clifford, S.Gadia, S.Jajodia, A. Segev, and R. Snodgrass (eds.). Temporal Databases：Theory, Design, and Implementation. Benjamin/Cummings Publishing Company, 1993.

[3] Güting，R. H. An introduction to spatial database systems. The International Journal on Very Large Data Bases(VLDB), Special Issue on Spatial Database Systems, 1994, 3(4):357~399.

[4] FrankA, Güting，R. H，J.C. Chorochronos: a research network for spatio-temporal database systems. ACM SIGMOD Record, 1999, 28(3):12~21.

[5] Jose ANTONIO COTELO LEMA，Güting，R. H. Algorithms for moving objects databases. The Computer Journal，Vol.46，2003.

[6] Wolfson，O. Moving objects information management：The database challenge. In：Proceedings of the 5th Workshop on Next Generation Information Technologies and Systems. Caesarea，Israel. 2000：75~89.

[7] O. Wolfson，S. Chamberlain. Moving objects databases：Issues and solutions. In：Proceedings of the 10th International Conference on Scientific and Statistical Database Management. Capri，Italy. 1998：111~122.

[8] Proser，D. C. S. Jesen. Indexing of network constrained moving objects. In：Proceeding of the 11th ACM International Symposium on Advances in Geographic Information Systems Database. New Orleans，Louisiana，USA. 2003：25~32.

[9] Wolfson. DOMINO：Databased for Moving Objects tracking. In：Proceedings of 1999 ACM SIGMOD International Conference on Management of Data. 1999：547~549.

[10]N. Pelekis，B. Theodoullidis，I. Kopanakis，and Y. Theodoridis. Literature Review of Spatio-Temporal Database Models. Knowledge Engineering Review，2005.

[11]Kanellakis P C，KuperG M，Revesz P Z. Constraint query languages. In：Proc of 9th ACM Symp on Principles of Database System (PODS90). New York：ACM Press，1990：299~313.

[12]Brodsky A，Segal V E. The C^3 constraint object-oriented database system：An overview. In：LNCS 1191. Berlin；Springer，1997：134~159.

[13]Chomicki，J. P. Z. Revesz. Constraint-based interoperability of spatiotemporal databases. In：

Proceedings of the 5th International Symposium on Advances in Spatial Databases(SSD'97). Berlin, Germany. 1997: Springer-Verlag. 142~161.

[14]Grumbach, S. P, Rigaux, L. Segoufin. The DEDALE system for complex spatial queries. In: Proceedings of 1998 ACM SIGMOD International Conference on Management of Data(SIGMOD'98). Seattle, Washington, USA, 1998: ACM Press. 213~224.

[15]Grumbach, S. P, Rigaux, L. Segoufin. Spatio-temporal data handling with constraints. In: Proceedings of the 6th International Symposium on Advances in Geographic Information Systems(ACMGIS'98). Washington, DC, USA. 1998: ACM Press. 106~118.

[16]Güting, R. H and Schneider M. Realms: A foundation for Spatial Data Type in Database Systems, Proc. 3rd Intl. Symposium on Larger Spatial Databases, Singapore, 1993: 14~35.

[17]Güting, R. H and Schneider M. Realm-Based Spatial Data Types: The Rose Algebra, Fern University Hagen, Informatic-Report 141, 1993.

[18]Güting, R. H, etal. Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Types, Proc. 4th Intl. Symposium on Large Spatial Databases, Portland, 1995: 216~239.

[19]秦小麟. 空间分析数据的研究方法与技术. 中国图象图形学报, 2000, (09): 711~715.

[20]秦小麟. 移动空间数据类型和操作的初步研究. 计算机科学, 2000, 27(01): 75~80.

[21]文韬. 空间数据库 NHSpatial 关键技术的研究与实现, [硕士学位论文]. 南京: 南京航空航天大学, 2006.

[22]Erwig, M., R. H. Güting, M. Schneider. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. GeoInformatica, 1999, 3(3): 269~296.

[23]萨师煊, 王珊, 数据库系统概论(第三版), 北京: 高等教育出版社, 2000 .

[24]Parent, C., S. Spaccapietra. Spatio-temporal conceptual models: Data structures, space, time. In: Proceedings of the 7th International Symposium on Advances in Geographic Information Systems(ACMGIS'99). Kansas City, Missouri, USA. 1999: ACM Press. 26~33.

[25]Erwig, M., R. H. Güting, M. Schneider. Abstract and discrete modeling of spatio-temporal data types. In: Proceedings of the 6th International Symposium on Advances in Geographic Information Systems(ACMGIS'98). Washington, DC, USA. 1998: ACM Press. 131~136.

[26]Erwig. M, M. Schneider. Developments in spatio-temporal query languages. In: Proceedings of the 10th Workshop on Database and Expert Systems Applications(DEXA'99). Florence, Italy. 1999: IEEEComputer Society.441~449.

[27]Kim, D. H., K. H. Ryu, H. S. Kim . A spatiotemporal database model and query language. The

Journal of Systems and Software, 2000, 55: 129~149.

[28]Kuroki. S, K. Lshizuka, A. Makinouchi. Towards a Spatiotemporal OQL for Four Dimensional Spatial Database System. Hawks.in: Proceeding of 8th DEXA.1997: 142~147.

[29]Theodoridis, Y., M. Vazirgiannis. T. K. Sellis. Spatio-Temporal Indexing for Large Multimedia Applications.in:ICMCS. 1996: 441~448.

[30]X.Xu, J. Han, W. Lu. RTtree: An Improved R-tree Index Structure for Spatiotemporal Database. In: Proceedings of the 4th International Symposium on Spatial Data Handling. 1990.

[31]Y.Thedoridis, TK. Sellis, A. Papadopoulos. Specifications for Efficient Indexing in Spatiotemporal Databases . in SSDBM. 1998: 123~132.

[32]Nascimento, M. J. Silva. Towards Historical R-trees. In: ACM SAC.1998: 235~240.

[33]Guttman, A. R-trees: A Dynamic Index Structure for Spatial Searching. In Proceedings of ACM SIGMOD Conference. 1984: 47~57.

[34]K. T, J. C. S, B. M. Layered implementation of temporal DBMSs-Concepts and techniques. In: Proceedings of the 5th International Conference on Database Systems for Advanced Applications. Melbourne, Vic., Australia. Singapore: World Scientific. 1997: 371~380.

[35]K. T, J. C. S, S. R. T. Stratum approaches to temporal DBMS implementation. In: Proceedings of International Database Engineering and Applications Symposium. Los Alamitos, CA, USA. 1998: IEEE Comput. Soc. 4~13.

[36]Bohlen, M. H. Temporal database System implementation. [J] SIGMOD Record, 1995, 24(4): 53~60.

[37]金培权，柳建平，赵振西．一种基于对象关系模型的时空数据库管理系统体系结构．小型微型计算机系统，2004，25(1)：108~111.

[38]Güting, R. H, M. H. Bohlen, M. Erwig, etal. A foundation for representing and querying moving objects. ACM Transactions on Database Systems, 2000, 25(1): 1~42.

[39]Forlizzi, L., Güting, R. H, Nardelli, E. and Schneider, M. A data model and data structures for moving objects databases. In Proc. ACM SIGMOD Intl. Conf. On Management of Data, 2000: 319~330.

[40]A. P. Sistla, O. Wolfson. Modeling and Querying Moving Objects. In Int. Conf. on Data Engineering(ICDE), 1997: 422~432.

[41]X. Meng and H. Sun. Modeling and Predictiong Future Trajectory of Moving Objects in a Constrained Network. In Int. Conf. On Mobile Data Management(MDM), page156, 2006.

[42]R. Praing and M. Schneider. A Universan Abstract Model for Future Movements of Moving

Objects. In AGILE Int. Conf. On Geographical Information Systems, 2007.

[43]Nicolai M. Josuttis. The C++ Standard Library: A Tutorial and Reference.

[44]Tore Risch , AMOSII Release 7 User's Mannual , 2006 , http://user.it.uu.se/~udbl/amos/doc/amos_users_guide.html.

[45]Gustav Fahl and Tore Risch, AMOSII Introduction. Uppasala Database Laboratory, Department of Information Science, Uppsala University . Sweden, 2000.

[46]Tore Risch, AMOSII External Interfaces. Uppasala Database Laboratory, Department of Information Science , Uppsala University Sweden, 2000.

[47] Daniel Elin, Tore Risch . AMOSII Java Interface. Uppsala Database Laboratory, Department of Information Science Uppsala University Sweden.

[48]X Zhou. Parallel Processing in Relational Database System, [Phd thesis]. University of Queensland, 1994.

[49]张林波，迟学斌，莫则尧. 并行计算机导论. 北京：清华大学出版社，2006.

[50]段安利. 空间拓扑分析操作的并行处理技术研究，[硕士学位论文]. 南京：南京航空航天大学，2009年.

[51]M. Erwing and M. Schneider. Visual Specification of Spatio-Temporal Developments. In 15th IEEE Symp. on Visual Languages(VL), 1999: 187~188.

[52]M. Erwing and M. Schneider. Query-By-Trace: Visual Predicate Specification in Spatio-Temporal Databases. In The Fifth Working Conference on Visual Database Systems, volume 168, 2000: 199~218.

致谢

在举笔即将完成这篇文章的时候，我猛然发觉自己即将离开校园，人生又将写下新的篇章。经历了找工作的喧嚣与坎坷，我深深体会到了写作论文的那份宁静与思考。在这离别之际，尽管依依不舍，却很珍惜，因为在我的生命中有那么多人和事值得感激。

首先感谢南京航空航天大学给我机会，我本愚钝，却有幸跻身于如此圣洁的科研殿堂，让我亲身体会了南航的良好科研氛围。教育的浩瀚知识，讲学的唯实，科研的创新将让我终身难忘。在本文完成之际，谨向母校致以最真诚的谢意和良好的祝愿，祝愿母校早日建成一流的研究型大学。

感谢恩师秦小麟教授，这篇论文是在他精心指导下完成的。从论文的选题、开题、文章结构的构筑，到最后的撰写定稿，每一个细节都得到了秦老师的细心指导与提携。导师严谨治学的作风和诲人不倦的态度将影响我的一生，给我以后的工作和学习指明方向。导师的创新思维使我无时不感受到挖掘与开拓的魅力。还要感谢秦老师给我参与“空间数据库管理系统及应用技术”项目的机会，这个过程让我学到了很多课本上学不到的知识，在此谨向我最敬重的秦老师致以最诚挚的谢意！同时也向在我学习期间给予指导和帮助的王立松副教授，皮德常教授，朱广蔚老师致以最衷心的感谢。此刻，似乎无法用言语来形容我的感激之情，惟愿师生情谊一生延续。祝愿老师及家人健康快乐，永远幸福。

感谢瑞典 Uppsala 大学 Tore Risch 教授通过邮件给予我无私的帮助，让我对 AMOSII 有了更深入的理解；感谢教研室已毕业的文韬师兄对我的帮助，他设计的 NHSpatial 空间数据库管理系统为我的研究工作和本文的形成指明了方向，消化吸收他设计的原型系统那段日子是我研究生期间最充实和最有所收获的时光。感谢他直接和间接的给我的这些帮助。

感谢实验室的兄弟姐妹们，他们对科研的热忱激励着我，与他们进行学术和技术讨论让我深受启发。他们是：杨丰，马忠成，吴浩，黄灿，储惠，计晓斐博士，陈亦菲博士，刘亮博士，戴华博士。特别是杨丰与马忠成，在我做论文和找工作过程中给予我无私的帮助。感谢师弟徐广龙，花费宝贵的时间对论文的内容提出了宝贵的意见。感谢携手共走的同窗好友，使得我的求学之路变得充实而又充满欢愉。感谢 S07043 班所有同学，是他们的科研精神和勤奋好学的高尚品质，感化和鞭策我不断进步。虽然相聚匆匆，但友谊天长地久。

回首往事，历历在目。老师的淳淳教导，同窗的真诚关心，同学的热情帮助，家人二十多年来在物质上对我学业上的无私支持，都使我不能忘怀，他们将永远激励着我不断前进。

最后衷心感谢审阅老师和答辩委员，你们在百忙之中抽出时间审阅论文和参加我的硕士答辩，为我指点迷津，谢谢您。

在学期间的研究成果及发表的学术论文

攻读硕士学位期间发表（录用）论文情况

[1] 段海亮，秦小麟，时空数据库体系结构与数据建模研究，2009年全国计算机技术应用与高等学校计算机教育论文集，第17卷，2009，已发表.

攻读硕士学位期间参加科研项目情况

[1] 鉴定项目："空间数据库管理系统及应用技术"，2008年获国防科学技术二等奖.

[2] 国家863高技术研究发展计划项目(2007AA01Z404)：基于网格的数据可靠存储与容侵关键技术.