SECONDO: A Database Management System for Moving Objects

Ralf Hartmut Güting
Fernuniversität Hagen, Germany
Outline

1. Moving Objects Databases
2. Spatio-Temporal Data Types
3. SECONDO Overview
4. SECONDO Demo
Moving Objects Databases

Goal:

• any kind of moving entity (time dependent geometry) can be represented in a database
• powerful query languages available to formulate any kind of questions about such movements

Two perspectives:

• location management
• spatio-temporal data

The Location Management Perspective

Problem: Manage the positions of a set of moving entities in a database, e.g. taxi-cabs in a city.

taxi-cabs (id: int, pos: point)

Unpleasant trade-off:

• frequent updates: high communication cost
• rare updates: high location error
**Moving Objects Databases**

Frequent updates not feasible for a large set of objects.

**Idea:**
- Store in the database not the position, but a *motion vector*. Describe position as a function of time.
- Perform updates only when the deviation between expected position and real position exceeds a threshold.
- Updates of motion vectors needed only much less frequently than position updates.

**Goal in location management:**
- Maintain dynamically the locations of a set of currently moving objects
- be able to ask queries about current positions, positions in the near future, or any kind of relationships developing between moving and static objects in the next time

**Note:** Not a temporal database at all; snapshot database maintaining the current state of the world. No history kept.
Moving Objects Databases

The Spatio-Temporal Data Perspective

- Consider what is stored in a (static) spatial database.
- Obviously all this may change over time.
- Keep track of history.

Two questions:
- What kind of data are stored in spatial databases
- What kinds of changes may occur?

Kinds of data: point, line, region; partition, network, ...

Kinds of changes: discrete / continuous

Discrete changes for any kind of spatial data. Already handled in classical spatio-temporal DB research.

Continuous changes: most relevant for point and region. Two basic abstractions: moving point and moving region.
Spatio-Temporal Data Types

Approach:

- Extend the strategy used in spatial databases to offer abstract data types with suitable operations.
- Offer spatio-temporal data types such as moving point (mpoint) and moving region (mregion)
- Values of such types are functions from time into the domains, e.g.
  - $f: \text{instant} \rightarrow \text{point}$ (value of mpoint)
  - $f: \text{instant} \rightarrow \text{region}$ (value of mregion)
Spatio-Temporal Data Types

Can describe discretely changing regions:

```
real_estate (owner: char(30), area: mregion)
```

But can also describe truly continuous changes:

```
flight (id: string, from: string, to: string, route: mpoint)
weather (id: string, kind: string, area: mregion)
```

The data types include suitable operations such as:

- intersection: $\text{mpoint} \times \text{mregion} \rightarrow \text{mpoint}$
- distance: $\text{mpoint} \times \text{mpoint} \rightarrow \text{mreal}$
- trajectory: $\text{mpoint} \rightarrow \text{line}$
- deftime: $\text{mpoint} \rightarrow \text{periods}$
- length: $\text{line} \rightarrow \text{real}$
- min: $\text{mreal} \rightarrow \text{real}$
Spatio-Temporal Data Types

Some Example Queries

Query 1: “Find all flights from Düsseldorf that are longer than 5000 kms.”

```sql
SELECT id
FROM flights
WHERE from = 'DUS' AND length(trajectory(route)) > 5000
```

*trajectory: mpoint -> line
length: line -> real*

Query 2: “Retrieve any pairs of airplanes that during their flight came closer to each other than 500 meters!”

```sql
SELECT f.id, g.id
FROM flights AS f, flights AS g
WHERE f.id <> g.id AND min(distance(f.route, g.route)) < 0.5
```

*distance: mpoint x mpoint -> mreal
min: mreal -> real*
Spatio-Temporal Data Types

Query 3: “At what times was flight BA488 within the snow storm with id S16?”

```
SELECT deftime(intersection(f.route, w.area))
FROM flights AS f, weather AS w
WHERE f.id = 'BA488' AND w.id = 'S16'
```

intersection: mpoint x mregion -> mpoint

deftime: mpoint -> periods

Approach described in:


Careful design of data types and operations can be found in:


See also the book:

Spatio-Temporal Data Types

Data Types

Spatial Types

point

points

line

region
Spatio-Temporal Data Types

instant

all temporal types for these

int
real
bool
string
point
points
line
region

all projections to range

moving(int)
moving(real)
moving(bool)
moving(string)
moving(point)
moving(points)
moving(line)
moving(region)

range(int)
rangereal)
rang(bool)
rang(string)

points
line
region

all projections to domain

periods
Spatio-Temporal Data Types

Abstract Model: Data types can be defined in terms of infinite sets.

Discrete Model: Data types must have finite representations.

Representation of types $\text{moving}(\alpha)$: Represent the temporal development of the value of type $a$ by decomposing the time dimension into a set of disjoint time intervals ("slices") such that within each slice the development can be described by some "simple" function. Called the sliced representation.

General strategy: Implement an algebra as a data blade, cartridge, extender, etc. in suitable extensible DBMS environments.
**SECONDO Overview**

An environment for implementing DBMS with new kinds of data models, suitable for research prototyping and teaching. Developed in the last ten years or so at University of Hagen, Germany.

- no fixed data model
- system frame can be filled with implementations of different data models, e.g.
  - relational
  - object-relational
  - graph/network-oriented
  - sequence-oriented
- goes beyond extensibility just by attribute data types
- system frame contains data model independent parts of a DBMS
- data model dependent parts implemented in algebra modules
- current “contents”: basically a relational system with several advanced data type packages

Open source software, available at

http://www.informatik.fernuni-hagen.de/import/pi4/Secondo.html/
SECONDO Overview

Three major components:

• **SECONDO kernel**:  
  - implements specific data models  
  - extensible by algebra modules  
  - provides query processing over the implemented algebras  
  - implemented on top of BerkeleyDB storage manager  
  - written in C++

• **Optimizer**:  
  - core capability: conjunctive query optimization  
  - currently supports a relational model with an SQL-like language  
  - written in PROLOG

• **GUI**:  
  - extensible interface for an extensible DBMS like SECONDO  
  - extensible by viewers  
  - sophisticated spatial / spatio-temporal viewer, extensible by data types  
  - written in Java
Components work together:

- GUI sends executable query (query plan) to the kernel, displays result
- GUI sends query to optimizer, receives plan, sends plan to kernel, displays result
- optimizer sends commands and executable queries to kernel to get information about DB objects, e.g. selectivities
SECONDO Overview

The SECONDO Kernel

Rough architecture:

- Command Manager
- Query Processor & Catalog: \( \text{Alg}_1, \text{Alg}_2, \ldots, \text{Alg}_n \)
- Storage Manager & Tools
SECONDO Overview

The SECONDO Kernel - Commands

A database is a pair \((T, O)\) where \(T\) is a finite set of named types and \(O\) a finite set of named objects. Seven basic commands to manipulate a database:

- `type <identifier> = <type expression>`
- `delete type <identifier>`
- `create <identifier> : <type expression>`
- `update <identifier> := <value expression>`
- `let <identifier> = <value expression>`
- `delete <identifier>`
- `query <value expression>`

Further commands for

- creating, deleting databases
- inquire about algebras, type constructors, operations, as well as types and objects in a database
- importing, exporting objects and databases
- transaction management
**SECONDO Overview**

**The SECONDO Kernel - Algebra Modules**

An algebra consists of

- a set of **type constructors**
- a set of **operations**

The module contains:

- **type constructor**
- **operator**
- **representation**
- **data structure**
- **data structure**
- **Create/Delete**
- **Open/Close**
- **Save/Clone**
- **In/Out**
- **KindCheck**
- **TransformType**
- **Select**
- **Evaluate**
**SECONDO Overview**

**The SECONDO Kernel - Algebra Modules**

Some currently available algebra modules:

<table>
<thead>
<tr>
<th>Algebra Module</th>
<th>Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>StandardAlgebra</td>
<td>int, real, string, bool</td>
</tr>
<tr>
<td>RelationAlgebra</td>
<td>rel, tuple</td>
</tr>
<tr>
<td>BTreeAlgebra</td>
<td>btree</td>
</tr>
<tr>
<td>RTreeAlgebra</td>
<td>rtree</td>
</tr>
<tr>
<td>SpatialAlgebra</td>
<td>point, points, line, region</td>
</tr>
<tr>
<td>DateTimeAlgebra</td>
<td>instant, duration</td>
</tr>
<tr>
<td>TemporalAlgebra</td>
<td>periods, rangeint, ..., mint, mreal, mpoint, mgpoint, mgline</td>
</tr>
<tr>
<td>NetworkAlgebra</td>
<td>network, gpoint, gline</td>
</tr>
</tbody>
</table>
SECONDO Demo