

# Database Tuning, ITU, Spring 2007

Rasmus Pagh

April 17, 2007

## Project – deliverable 4

**Deadline (strict): May 3, 23.59 Danish time.**

## Time plan for deliverable 4

**April 17.** Description, question/answer session.

**April 24 and May 1.** Office hours for project supervisor (Milan Ruzic).

**May 3.** Hand-in by e-mail (pdf file) to `milan@itu.dk`

**May 8.** Feedback meetings.

## Purpose

The purpose of the practical part of this deliverable is to gain experience with concurrency control mechanisms and applying methods for tuning transactions. The purpose of the theoretical part is to train your analytical skills, based on knowledge of temporal and spatial databases.

## Overview

In this deliverable you continue your work on the VXL database. This time emphasis is on handling many concurrent transactions, some of which involve updates. You should use your knowledge of concurrency control, and consider the recommendations for tuning transactions given in the course.

We provide you with a program skeleton for issuing concurrent transactions to the database. You will be working with several transactions, specified below.

You can download the new skeleton program from the ITU file system: `H:/milan/DBT-files` or `/import/home/milan/DBT-files`. Milan will be presenting it on April 17, and is the person to ask questions regarding these programs.

## Transactions

19. Insert the position (x,y) of given vehicle at given time.  
`public void transInsertPosition(...)`
20. Find the vehicle closest to position (x,y) at given time t. For any vehicle, its position at time t is determined by the last GPS entry with timestamp not later than t.  
`public int queryClosestVehicle(...)`
21. Find a vehicle that can be booked between time t1 and t2, and whose driver is based in a given county.  
`public int queryAvailableVehicles(...)`

22. Book a given vehicle between time t1 and time t2, checking that it is indeed available. Unspecified fields from BookableSchedule should be set to null.  
public void transBookVehicle(...)
23. Book all available vehicles with storage capacity of at least c kilograms between time t1 and t2.  
public void transBookStrongVehicle(...)
24. Book all available vehicles with storage capacity of at least l liters between time x and y.  
public void transBookBigVehicle(...)
25. Remove from the database all information regarding a given parcel.  
public void transRemoveParcelHistory(...)
26. County c now belongs to region r. Update the database to reflect this fact.  
public void transChangeRegion(...)

## Theoretical investigation — Spatial and temporal indexes

1. **Uniform grid file:** Let's call a grid file *uniform* if all the grid rectangles are squares of the same size. Consider a uniform grid file, based on the GPS coordinates, for road segment endpoints. Suppose we want to support nearest neighbor queries of the form: Given the GPS coordinates of a point, find the road segment endpoint closest to it. Would you consider storing a uniform grid file to support these queries, as opposed to traditional indexes (B-trees or hash files)?

For the VXL geographical data, estimate the size of this grid file (i.e., how many grid squares are needed) if we want that each square has at most 1000 points in it (assuming they fit in a single block). This should be done by issuing suitable queries on the geographical data. (**Hint:** For different square sizes, write an SQL query that groups the endpoints according to what square they are in, and returns the maximum size of such a group. Use binary search to find (roughly) the biggest possible square size. Finally, write a query to compute the number of nonempty squares.) Compare the grid file size with the size of a traditional index. Should VXL buy the uniform grid file supplied by BlastSoft AB?

2. In the VXL database, identify all the relations which would benefit by making them temporal relations, with corresponding persistent indexes. Describe the some sample temporal queries one could perform on those relations. (You don't need to be exhaustive.)

For example, one can make the "customer" relation temporal by storing the joining and leaving times of customers, and listing all the customers that are "active" at a given time can be efficiently answered using a persistent B-tree.

## To be handed in

A pdf file with the following:

- (Optional.) Description of things that were changed since the third deliverable (data model, queries, indexes).
- A description of how each transaction was implemented. This could be pure SQL, PL/SQL, or a combination of Java and SQL.
- The output of the test program.
- A description of the reasoning behind the choices made for each transaction. Were other options tried, and with what results?
- Answers to the theoretical investigations.