Database Tuning, ITU, Spring 2008

Rasmus Pagh

February 5, 2008

Project – deliverable 1

This is the first part of the Database Tuning project. The project is to be carried out in groups of 3-4 students. It is up to the members of your group to agree on how to work together (meeting times, etc.) The project will be structured in 5 parts, where the first 4 have a deliverable to be handed in during the course. The final project report will contain revised versions of these deliverables, with the possibility to work on and write about additional (performance) aspects of the development project.

Deadline for deliverable 1: February 21, 23.59 Danish time.

Purpose

The Database Tuning project is a database development project with focus on performance. The performance will be tested by programs that simulate use of the database, provided by the teachers of the course. The first deliverable builds upon skills acquired in an introductory database course, and aims at getting an understanding of the problem domain addressed by the database by constructing an initial E/R diagram, a corresponding relational data model, creating the relations in Oracle, and finally loading data into the database.

Time plan for deliverable 1

February 5. Project introduction. Case description, question/answer session.

February 12 and 19. Office hours for project supervisor (Milan Ružić).

February 21. Hand-in by e-mail (pdf file) to `milan@itu.dk`.

February 26. Feedback (individually for each group).

Time plan for future deliverables

Deliverable 2: Start February 26, **hand-in March 20**, feedback around March 27.

Deliverable 3: Start March 25, **hand-in April 10**, feedback April 15.

Deliverable 4: Start April 15, **hand-in May 1**, feedback May 6.

Extras + final report: Start May 6, **hand-in May 21**, feedback at exam!

Case description

The following is only a partial description of the case. To obtain a full understanding it is probably necessary to ask questions to the recipients of the result of the project (your teachers).

Viking Xpress Logistics is a newly started Danish delivery service. Its customers order door-to-door delivery of parcels nationwide. The business idea of VXL is to provide a very flexible service that will allow customers to hand over their post at any time, and still ensure that parcels are routed in a fast and cost-effective way to its destination. The routing of parcels will be handled by software by the little-known German company *Pfadfinder AG*. The aim of this project is to create a database that will allow VXL to keep track of their parcels and other business/organizatorial information.

Traditional logistics companies gather all post in large hubs on a daily basis, and ship everything from there. The idea behind VXL is to provide a more flexible (and often faster) routing of parcels by using an infrastructure similar to public transportation. For example, a parcel traveling from the IT University of Copenhagen to Kronborg Castle in Elsinore might first be shipped by car to Taarnby Station, then later picked up and transported by train to Elsinore train station, and after a short waiting time finally be shipped to Hamlet's castle by a bike courier. To make this work, VXL has safe storage containers set up at railroad stations and thousands of other places across the country, where the parcels can be placed while waiting to be carried further. Some VXL vehicles travel according to a fixed schedule, and some can be reserved to carry a particular parcel (mainly at route end points). VXL's customers have client software installed that makes queries to the database on routing possibilities, and reserves space on vehicles for a particular parcel. Pricing may vary according to what route is chosen.

The VXL database must contain:

- All road segments in Denmark, including name of the street, postal code, and ranges of street numbers for left and right side of the street. One street may span over several road segments.
- All endpoints of road segments, including their geographical coordinates. Several road segments may share an endpoint, and each segment has exactly two endpoints.
- Name and address (road segment and street number) of each customer that has ever used VXL service. More than one customer may reside on one address.
- For every vehicle, the type (train, truck,...) and model name is stored, along with information on date of purchase, maximum speed, carrying capacity (both in kilograms and liters), and mileage. Milage is updated weekly.
- Name and ID of every driver. Working hours are recorded for each driver in a weekly schedule. For every day of the week the work shift of the driver (e.g. 8-16) and the vehicle he operates are recorded. During a work shift, no more than one vehicle is assigned to each driver. Further, each vehicle has a single driver during a work shift.
- The drivers are organized by county, and the counties are organized by region. It must be possible to identify the drivers that are based in a particular county or region. (There is no requirement to keep track of the county that a vehicle is in.)
- For each safe storage container, its address, capacity (kilograms and liters), the name of the vendor that built it, and the date it was put to use.
- The driving schedule of every vehicle. Some vehicles have a fixed weekly schedule, while others are bookable by the Pfadfinder software. (This information is stored in the database.) Vehicles that run on the fixed weekly schedule connect two storage containers, while vehicles on specially booked routes connect a storage container and a custom address and then return to the container. In both cases, no stops are made in between. For fixed routes, start and arrival time are listed, while for custom routes start and return time are stated.

- Information about every parcel, including the customer who ordered the shipping, shipping price, time of order, pickup and delivery address (road segment and street number), and also routing information (travel plan) determined by Pfadfinder software. Travel plan contains a list of references to vehicle schedule entries, and planned arrival time to destination.
- A complete history of all things that happen to parcels, such that customers can check the locations and progress of their parcels at any time. This includes depositing in and pickup from safe storage containers (container identifier and a timestamp). Each action involving a parcel is marked by an identifier of the VXL driver responsible. Also, the time (in seconds) used on each action is recorded.
- A GPS based system informs the VXL headquarters about the position of each of its vehicles every minute. The geographical position is recorded along with the corresponding timestamp. No positioning information is maintained when vehicles are idle on their endpoint stations.

The database must ensure referential integrity for foreign key references. However, other constraints such as the fact that a driver cannot be driving two vehicles at the same time are supposed to be enforced by the software that adds data to the database.

Implementation and performance testing framework

The Java SE platform will be used to create an application that interacts with the database you create and manage. This Java application has two purposes ?– to load the supplied test data into the database, and to carry out performance tests of queries that will be assigned in the project.

A considerable part of the implementation is provided to the students, and will be common for everyone. This general part of the code sets out a framework of interaction with the database, but it contains few statements of *direct* communication with the database. Each group will probably make a different database schema, so it will be your task to fill in `jdbc` statements that implement the required queries into your database.

The program is organized into three packages — `controllers`, `dbapp`, and `loaders`. The package `controllers` contains two classes: DataLoader and QueryTestUnit. These are the only two classes in the Java project that contain the method `main`, and thus can be used as starting points of Java applications. The structure of the DataLoader class is quite simple, comprising of statements that make calls to methods in the package `loaders`. The class QueryTestUnit acts as a test driver. Each query type assigned during the project will have a corresponding method (in QueryTestUnit) that generates test parameters, or sometimes loads them from the data files. Such a method makes calls to the procedure that implements the query. The calls are made a number of times, each time with different parameters, and the total execution time for that sequence of queries is output to the user. Sometimes queries will be mixed with SQL update statements, and sometimes they will be called from concurrently running threads. Your main involvement with the code of these two classes will be to disable procedure calls in the `main` methods by turning them into comments, or uncommenting them, depending what parts you want to execute at the moment.

The package `dbapp` contains the class DatabaseInterface, which will be the focus of your work in the first Java application you make. Initially it contains skeletons of methods that need to be filled in with `jdbc` statements. These methods are called from the other two packages. On the other hand, this class invokes no methods other than those in `java.sql` package (therefore it would be a leaf class in the class diagram). Comments in the file DatabaseInterface.java, along with parameter declarations, provide additional details about domains of relation attributes.

The package `loaders` contains classes that perform loading of the supplied data. It is not expected that students will need to make any changes in these files. One class may load data that correspond to more than one relation. **Tip:** You may want to disable integrity checks while loading to speed up the process.

You can download the above packages from the ITU file system: `H:/milan/DBT-files` or `/import/home/milan/DBT-files`. They will be updated and extended for future deliverables.

The JDK compliance level of the compiler should be set to 5.0 (a.k.a. 1.5). It is advised that you use Eclipse SDK. Oracle jdbc drivers need to be added to the build path (or CLASSPATH environment variable). The drivers are located in the directory [`ORACLE_HOME`]`/jdbc/lib/`, which for example may be `C:\OracleXE\app\oracle\product\10.2.0\server\jdbc\lib`. The file `ojdbc14.jar` has to be included in the build path, but it might be good to also include `ojdbc14_g.jar` because of additional debugging features.

To be handed in

A pdf file with the following:

- An E/R diagram for the VXL database, plus written explanation where necessary. You must state what E/R notation is used (preferably Chen or the one in RG).
- DDL statements for creating the corresponding relations in Oracle, including integrity constraints. For more detailed information on data types (e.g. maximal length of strings), see the comments and declarations in the java program that loads data.
- Documentation that you loaded the data correctly, e.g., queries on the size of each relation, with query results.

Group work contract

Many of you will be working with a group of people that you did not know before. We encourage you to discuss, informally, a *contract* about the group work, since this is a good way to adjust expectations and limit frustration. (In extreme cases you may find out that you are simply not compatible — in that case, contact your teachers.) You may, for example, discuss:

- Do we want a stable work load, or is it better to do everything in an intense period before the deadline?
- Should we work closely together, or should we try to split the work as much as possible?
- When should we meet? How should we coordinate our work besides meetings?
- How much time per week can we allocate to the project?
- What is the level of ambition? (Note that in the final report it is possible to do individual sections, so the level of ambition with respect to grades does not need to be exactly the same for the whole group.)

Example queries

Some questions about the data may be answered by looking at the java program that loads data. Note, however, that it will not generate the full data set yet. To further aid your understanding, here are some example queries that we want the database to support:

- Find all 5-way junctions in a given postal code.
- Find the addresses (street number, street name and postal code) of all customers whose name starts with the string s.
- Find the postal code with the highest number of customers, and list the names of all the customers with that postal code.

- Find the number of customers in the post codes between x and y.
- Find the number of parcels that have arrived at storage s on a given date.
- Find the first storage to which parcel p was sent.
- Find all the parcels that were first sent to storage s.
- Find all the storage locations in the rectangular region with diagonal coordinates $(x1, y1)$ and $(x2, y2)$.
- Find the IDs of all drivers who drive a vehicle of a given type.
- Find the names of all drivers in region *reg* who drive a vehicle with capacity more than *cap* on a given day of the week, according to the fixed schedule.
- Find the total number of hours worked per week by drivers in the region named `s`.
- Find the number of customers that has not sent a parcel after date $\hat{\text{d}}$.
- Find all parcels sent from postal code `p1` to postal code `p2`
- Find all vehicles that can carry more than 1000 kilograms, *and* have driven less than 50000 kilometers.
- Find the newest (most recently inserted) entry of the schedule of the bookable vehicle with id `vid`. This is the one with the highest value of schedule_id.
- Find all the vehicles the were bought before date `d` *or* have driven more than `k` kilometers.
- Find the name of every driver who picked up a parcel at the customer with the address `(sid,house-no)` on date `d`.
- Find, for every customer, the price and delivery address of the most expensive parcel sent. Report the result sorted by price in decreasing order.
- Find the total time spent on loading and unloading the parcel with ID `p`.