Assignment 3 - Database Tuning F2012

Theo Andersen - tha@itu.dk

May 30, 2012

1 Introduction

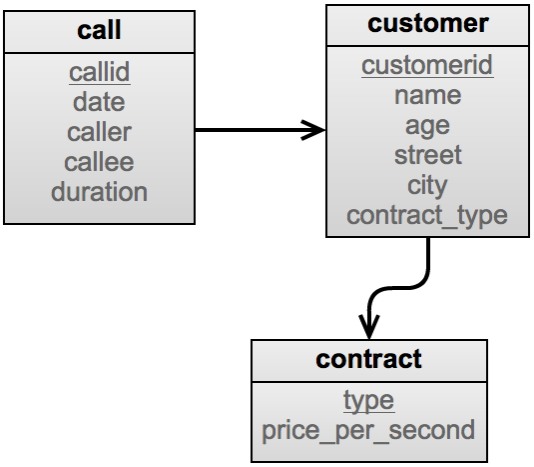
This paper is my answer for Assignment 3 in the course Database Tuning F2012 at ITU.

The assignment is to explain solutions to the tuning of a given query and a database schema, and to describe experiments to quantify the expected effects of the proposed solutions (It is not required to run them).

The domain of the exercise is a web-site where the customer can access the current state of the monthly phone bill.

1.1 The schema

The schema of the database described in the following diagram:



It is known that there are 100.000 rows in the customer table, which means that we can expect a fair amount more in the call table as it contains the calls for all customers. The contract table will propably be a lot smaller than customer as it describes the types of contracts for the customers.

1.2 The query

The query that is to be optimized is as follows:

```
select call.date, call.duration, call.duration * contract.price_per_second
from customer, call, contract
where customerid = ?
and customer.contract_type = contract.type
and call.caller = customer.customerid
and MONTH(call.date) = THIS_MONTH;
```

The query returns the date, duration and total cost of a phone call made by a given customer in a given month. It selects a given customer and creates inner joins on call and contract, having the customer as the caller of the call and the date of the call being within the current month

2 Tuning

The assignment text does not describe any indexes that might be created on the tables in the schema. If there are no indexes, the database will be forced to use table scans which are slow on larger tables (customer and especially the call table) and it wont be able to use indexes for better locking. Most of this solution will be focused on indexes.

2.1 Isolation level

Given that results of the query is meant to view the current status of the monthly phone bill, we do not need a very strict isolation level. The default read committed isolation with read committed snapshot enabled will be fine as the user does not need the exact correct data to the millisecond, but merely an overview of his current bill. This read committed snapshot will also ensure that any insert or update, wont lock a read on one of the tables.

2.2 Indexes

Customer

First a non-clustered index should be put on the customerid primary-key of the customer table. This would speed up the first look-up of the given customer in the customer table. A clustered index wouldn't be faster here, as this is only a point query. A hash-index might also be beneficial here, as the query is a point-query.

An experiment would be to test the response time and execution plan of the query with both no-index and the non-clustered. Because the table is pretty big i would definitely expect the non-clustered index to perform the best.

Contract

Contract is a small table and is accessed by multi-point query by the inner join, but i would still use a non-clustered index on it. This might not make it faster than a scan as there probably isn't many rows in the contract table, but an index on the primary key will help the database with better locks when it needs to update or insert. The price_per_second will also be included to this index, so that the index becomes covered and the database don't need to look-up the data for each hit. This index is not made clustered as we aren't performing any range queries on it, and thus wont have any benefit of the index.

The experiment here will be without and with the non-clustered index. A really small table will perform better with a table scan than an index, so this might be faster.

Call

The call table is the biggest of the three tables, meaning that it most certainly contains several million rows. Being that we statistically need under 1/100.000 of the data in the call table (its only for one customer), this would make a table scan a much slower result than using an index.

The initial reaction would be to use a non-clustered index on the caller and the date to make a covered index. The problem is though with the last filtering on the month of the date. If we're not careful, the database might not use that index, as it doesn't match the date to MONTH(date), and that would mean that we lost the use of the index on the caller as well, making it a full table scan on the largest of the tables. So the first experiment on this table would be to see if the database would choose the index, and to look at the execution plan to see how it would handle the MONTH(date) part.

Another way to look at it would be to change the query to use be a range query instead of the MONTH(date) thing, making the last line become this instead:

```
and call.date < NEXT_MONTH
and call.date > PREVIOS_MONTH
```

Now firstly this implies a slight change in the requirements of the query, which are explained in the later "Query might return calls for multiple years" section. But given that this change is okay, the changed filtering would mean that the index would most definitely be able to use the covered index including date as well.

Being that the date always is incremental, because its being inserted as calls are created, we could use this change to create a clustered index on date instead. A clustered index on date, would mean that it would utilize the clustered index to find the dates faster (as the data is then already

sorted by the date). So with a clustered index on date and a non-clustered on the caller, the database should be able to use the clustered index for the range-query on the date, and the non-clustered on the caller.

An experiment should be made to try out these two versions of the indexes. Also whether the database would first use the clustered index on date and then filter on the customer or the other way around, would depend on the statistics on the sizes of the tables.

I would expect that the first version wouldn't be able to use the covered non-clustered index, as it had to convert all the dates with the MONTH(date) first, in memory, and then filter on this. The second version should be faster as it is more clear that the new query could use the two indexes. But whether the database filters by date or by customerid first would be up to the statistics.

3 Problems with the schema and query

I have identified a few possible problems with the assignment, which are issues where the given database and query might give problems with the business case described in the assignment.

3.1 The query might return calls for multiple years

It seems that there is no filtering for which year the month should be shown. So the query will likely output every call done in a month, for every year the customer has calls in this database. This might not be the expected required functionality of the query, and the month-filtering should perhaps be tested to ensure that it returns the expected rows.

3.2 Why isn't the total call price summed on insert?

Looking at the schema it seems strange the the total call price isn't summed and inserted into the call table along with the duration. Having it summed when running the query, implies that the contract price_per_second can be changed, but this would be wrong for existing calls as it would change an existing calls price. This structure would make it possible for an old call to change price, when the company decides to change the contract pricing.

I would think that the more correct structure, would be to have a total_cost column on the call-table, and have inserts into call calculate this total cost from the given contract price_per_second at insert time.

Another way this could be modelled would be to decide that the contract price_per_second is never changed, but new contracts are added and then used whenever the contract pricing changes.