# Gentle introduction to optimization

## (Oracle-biased)

Vít Špinka

July 23, 2009

# Goals of this short workshop

- Go through the options you have (and can) consider when striving to develop a well-performant database-based application
- Don’t expect detailed cookbook
- In the printed form, it’s mainly a list of options and helps one to not forget they exist (or to hear about them for the first time)

# Basic rules

- Phases of optimization, in this order:
- Logical data model
- Application
- Physical data model
- Instance

- Early bird gets the worm – you can only patch the things you made wrong in previous phase

- It’s easy, but experience shows that it’s not matter-of-course
  - On a well designed system, tuning is easy and fun; on a bad system, it’s pain and sometimes even black magic. The difference is breathtaking.
  - Disadvantage: by tuning a well-designed system, you learn less☺

# Logical data model

- Two basic options to follow:
  - 3. normal form (OLTP, DWH)
  - Star-schema (datamart)
- Always define keys and relationships so they can be enforced, ideally directly by the database engine
  - If you really need, you can omit them from physical model

# Application

- Application “must make sense”
- Choose proper tools for the job
- Application tasks should be easily answerable from the data
  - Usual business operations work in a streamlined way – when we have good model for the business data, it’s easy to solve business tasks
  - Sophisticated tasks (scoring, etc.) will be of course always complicated
- Don’t bend, don’t abuse the model
  - You will do it in subsequent releases anyway…

# Application

- Best optimization is to do nothing
  - Cache the results
  - Materialized views
  - Limit frequent queries
- Partition load
  - Cluster – don’t let the nodes compete for the same data
  - Time – don’t let the users compete among themselves, or with batches
- Sometimes a trick can help
  - Just return OK to the user and do the actual work later (users asses response time, not the actual system load)
  - Or vice versa, prepare for the users beforehand (e.g. MVs at night)
- Optimizer / parse time
  - Binding is very good for frequent queries (usual OLTP system)

# Physical data model

- Help the optimizer
  - proper data types (dates as date, numbers as number)
  - not null
  - foreign keys
  - check constraints
  - use NULL, not arbitrary values
- Indexes, materialized views, partitioning
  - B*tree indexes are fundamental to OLTP
  - MVs help in datamart
  - Partitioning is handy in DWHs
  - But usually, you end up with all of them
- Keep an eye on High Watermark
- Use temporary tables (saves redo)

# Indexes

- Fundamental to database design – B*tree
  - Composite
  - Reverse
  - TDE
  - Index FFS
  - Index as a skinned-down version of the table
- Bitmap indexes
  - Bitmap join index
- Local/global

# Instance

- Up-to-date statistics!
  - DB objecs
    - Histograms are not trouble-free
    - History
    - Dynamic sampling / manual statistics (above all for temp tables)
  - System
  - Sys objects
- In DWH expect sorts and hash joins, i.e. PGA
- OLTP “runs on” cache, i.e. SGA
- 11g: extended statistics

# Tuning

- Set the goal first (the acceptable response times)
- Always solve the lowest-hanging fruit first
  - And then repeat the tests, solving one thing often solves some other too (or, sadly, makes them worse)
- There are always things that inherently must take long

# Who’s the culprit?

- It’s not always about the database
  - There is often a lot of components from database to database
- Instrument the code
  - It’s always handy if the application can itself identify the bottleneck
  - It’s proactive – you can monitor the system and action before the users complain
  - Sometimes the customer can use it to measure system availability (i.e. verify that the response time is acceptable)
  - The easiest way: set client_id and module/action, 10g can utilize it (but don’t let be satisfied with such a simple solution)

# Diagnostic tools

- Wait statistics
- osm (Craig Shallahamer)
- AWR, statspack
- Sql trace (tkprof, tvd$xtat - Antognini)

- explain plan, test runs
  - Watch out for environment differences! (user, NLS, CBO settings, ...)

- 10046 sql trace
- 10053 CBO
- 10132 dump (actual plan, outline, CBO settings)
- v$sql_plan

# Action tools (statement-level)

- Hints
- Outlines
  - Force one plan
- SQL profile (10g)
  - opt_estimate hint
  - Corrects CBO estimates so it can produce optimal plan
- SQL baseline (11g)
  - Prevents changing of a good, known existing plan
  - Includes plan evolve

# What to tune?

- See CPU+ wait statistics for the top event, it will show the way
  - db scattered read – full scans
  - db sequeuntial read – indexes
  - log switch – redo logs
  - enqueue – locks (watch out for ITLs)
  - free buffer – too many dirty blocks
  - buffer busy – block contention, identify type of block and object
- We need to identify load at that time, the culprit session
  - Events are not assigned to statements
  - Quite often, such assignments would not make sense system-wide (one bad query will make everyone look bad)
  - Use trace to identify waits pertaining to that session (in easy cases, you will directly see what to optimize just by finding the offending SQL)
- Application and OS statistics should support your guess

# Books to read

- Kyte: Effective Oracle by Design
- Lewis: Cost-Based Oracle Fundamentals: v. 1 (Expert's Voice in Oracle)
- Antognini: Troubleshooting Oracle Performance
- Milsap: Optimizing Oracle performance
- Oracle Wait Interface: A Practical Guide to Performance Diagnostics and Tuning