Beating the Oracle Optimizer – Jonathan Lewis

Class Agenda

08:00 – 08:30 Check-in. Complimentary breakfast served

08:30 – 10:00 **Single table access paths**

We start with a brief reminder of what we need to know to write efficient and scalable SQL, and then we examine the B-tree and bitmap access paths that are automatically available to Oracle as it accesses a single table before extending Oracle's strategies with a few manually constructed strategies that become available if we can rewrite the SQL.

10:00 – 10:30 Break – coffee and informal discussion

10:30 – 12:00 **The two-table join**

In this session we examine a very simple join and note a fundamental limitation in the optimizer's ability to find the best strategy for joining two tables. We see how we can overcome this limitation – at a cost of more complex SQL – and look at the way we need to think about joins to minimize the work we do, noting that the possible benefit isn't always as great as we might first think. We also review a method for keeping the code simpler at a risk of variable performance – then find we can eliminate the variability by again increasing the complexity.

12:00 – 12:45 Break – lunch and informal discussion

12:45 – 14:15 **Complex Joins**

After setting the groundwork with single table access paths and two table joins, we go on to more complex examples, showing how the principle can be used to emulate data warehouse patterns of query in a structure designed for OLTP data access; even to the extent of emulating a Star Transformation in Standard Edition Oracle where bitmap indexes are not implemented.

14:15 – 14:30 Break – coffee and informal discussion

14:30 – 16:00 **Structures and Features**

Falling back to slight more standard SQL, we take a look at the way in which we can use features like function-based indexes, virtual columns and deterministic functions in the newer versions of Oracle to reduce work. We also look at the ways in which structures such as sorted hash clusters and partitioning allow us to re-think the way we write SQL to minimize the work done.