The key to successful tuning with SQLT is to use SQLT regularly and for real-life problems. If you’ve reached the final chapter, you should now consider yourself a card-carrying member of the SQLT supporters club. You’ve learned a lot about what SQLT can do, and along the way you’ve probably learned some things about the cost-based optimizer and the Oracle engine. Let me remind you of some of the features we came across on our journey:

- The effect of statistics on execution plans
- The effect of skewness on execution plans
- How the optimizer transforms SQL during parsing
- How profiles can help you temporarily freeze an execution plan
- How adaptive cursor sharing works
- How dynamic sampling works
- How cardinality feedback works
- How you can use SQLT with Data Guard
- How test cases can be built with SQLT to allow exploration of the execution plans
- How to use the brute force of XPLORE to look for unexpected effects on the CBO from upgrades and other changes.
- How we can use the COMPARE method to investigate two SQLs
- Last but not least we talked about the health check script, which is a good second choice if SQLT is not available.

In this chapter I’ll try and give you a quick overview of a methodology I use to approach a tuning problem. Naturally all methodologies have exceptions, but it’s better to have a default plan than having to determine a new one for every occasion. I’ll also give my opinion as to why SQLT is the best tool available for tuning (apart from being free). Then I’ll discuss some platform issues and assure you that SQLT and the examples we’ve covered do not just work on one platform. Finally I’ll mention a few resources you should be aware of as you continue your greater journey into the world of tuning.

**Tuning Methodology**

Tuning methodology is not this book’s main theme, but I feel I need to say something about it because SQLT can be a central element to a good strategy. The lack of a central methodology to attack SQL tuning problems has always, in my opinion, been the main problem for DBAs especially but also for developers who have to create efficient code. When
you have a tuning problem, where do you start? Usually it all depends on what kind of problem you have. To help you here's my five-step method.

1. Get an AWR report for the problem time. If there is a "significant" problem in the “Top 5 waits” section of the report, deal with that first. If there is nothing obvious there then check the “SQL Report” section of the AWR report. If there is an SQL using more resources than other SQLs, then get an appropriate SQLT report. Always be led by the evidence presented in the AWR report and not by your own hunches or guesswork.

2. If it’s an individual SQL problem, start with SQLT XTRACT or XEXECUTE (depending on whether you can reliably run the SQL) and then use the information to go from there. If you can’t use SQLT, then use SQLHC.

3. Evaluate the information collected and scrutinize any information that looks out of the ordinary (this is where the constant practice helps, because you begin to recognize out-of-the-ordinary behavior on your system).

4. Investigate any anomalies and make sure you understand them. They may be benign. If the anomalies cannot be explained then try and assess if they could be the cause of your problems.

5. If you do end up investigating an individual SQL that has changed performance, remember there are many SQL tools that can be deployed to get more information. Build a test case and use COMPARE or use XPLORE in desperation (if you have the time).

This high-level methodology has a few key elements. The first is to recognize things that are out of the ordinary. To do this you must first understand what is normal, just like our alien visitor back in Chapter 2. The second element is the knowledge of how things work in the optimizer. This takes practice and some reading (hopefully this book helped).

It’s important to recognize that SQLT is not the first step in this methodology. Although SQLT is useful for many tuning problems, the first step should be to assess the overall system performance, which can be best done from an AWR report against the appropriate database. If the problem is related to the operating system, you may find your solution there and need never look at a SQLT report or indeed any SQL. If your problem is with the database then the AWR report is a good starting point for memory requirements, unexpected waits (seen in the top five events). See Figure 15-1, which shows the section of the AWR report with the top five waits on the system. This report is not very typical but at least shows no high percentages for unusual waits.

If your system is heavily loaded you should see waits in this section of the report that may require investigation. Depending on what these waits are, you may be led to the SQL section of the report, which then may suggest an investigation of a particular SQL or perhaps one or two. Then SQLT can be used to good effect (as long as the SQL is not some internal Oracle code). See below in Figure 15-2 for the top SQL sorted by buffer gets.