Preface

Someday people will query Oracle databases considerably differently than they do now. When that day arrives, expect to see vice-presidents, clerks, department managers, secretaries, and board members asking questions of organizational databases, receiving answers, and asking additional questions until they get their arms around whatever problem they currently confront. The central feature is that people without any database experience will directly and routinely use databases to address job-related questions. They will do so without any programming and without any intermediary to translate the ad hoc query into language understandable by the database. The process of asking ad hoc queries will approach the ease and convenience of speaking with another person. It's just that in this case the conversation will occur with a database containing information that can help solve day-to-day operational or longer-range policy and strategic issues.

It's helpful to have a vision of future ad hoc queries. But getting from the reality of today to the promise of that vision entails considerable work and great uncertainty as to how, exactly, we can move forward. Today, ad hoc queries are primitive by comparison with the scenario sketched above. Questions, of course, arise naturally in the course of a job. But most are triaged before they're even uttered aloud, simply because the person asking the question does not personally have the capability to query the database and because the process of actually getting an answer to the query involves so much effort. Normally, a query must be asked through proper channels, defended as to priority and importance, and queued with other queries to be programmed by information systems staff. Most questions simply wither rather than face this process; organizations suffer a hidden cost from questions unasked.

The present tools for ad hoc querying are also primitive compared with those which surely must evolve. Present query tools generally fall along a continuum. Programming tools exist at one end of the spectrum. These include SQL, SQL*PLUS, procedural extensions like PL/SQL, and procedural languages like SQR that provide embedded SQL. End-user tools dominate the other end of the continuum. Typically these tools are characterized by point-and-click, graphical interfaces that shield users from database complexity.

Both extremes exact penalties. SQL and procedural languages permit direct access to transactional databases, yet only programmers and analysts can invest the time required to use these tools proficiently. Organizations using these tools typically funnel ad hoc queries through intermediaries, who then write the query and return an answer. Much of the success in this approach depends on the skill of the intermediary — how well she knows the business rules and can interpret the original query correctly, how well she understands the database tables and their idiosyncrasies, how good a programmer she is, how persuasively she can present the findings, and how fast she can process the scores of questions queued for her.

The intermediary model is far from ideal. Not even the best of intermediaries can provide the real-time response needed for query, response, requery, response and so on. Intermediaries also can endanger an organization by creating points of brittleness. If nearly all queries of a particular type pass through one person, the organization becomes heavily dependent on that person. All it takes is a better job elsewhere and the conduit into the database suddenly disappears.

Graphical end-user tools solve some of these problems but create their own costs. These tools were developed with ease of use in mind. While many of the early efforts merely slapped a graphical interface over a programming language, several end-user query tools do now make data more accessible to nonprogrammers. The downside is that these tools cannot handle the complexity of fully normalized relational databases.

Perhaps an example is in order. One of us works for a college that uses an administrative database implemented with Oracle7 products. To comply with federal regulations, the college must distinguish new students from continuing students and, among new students, must distinguish freshmen from transfers. You might think these distinctions would be easy, but the federal definitions of transfer students actually make it difficult to identify these students until well into their first term of

enrollment. The query that makes this distinction uses simple but convoluted logic that need not be detailed here. Suffice it to say that the query requires about 50 lines of SQL, two correlated subqueries, one UNION, and a complex DECODE across correlating criteria. The question is simple, the SQL is not; but even this SQL is relatively modest compared with other queries.

End-user query tools are not equipped to handle complex queries or even simple queries against complex relational databases The solution generally has been to change the database — to create special data warehouses and extracted data tables intended only for querying. Data get copied, cleaned, reexpressed, combined, organized, filtered, moved, and inserted into the query-only data tables, which are then suitable for use with the end-user tools. For large warehouses, the expense can be enormous.

Clearly, something better must evolve, something that retains and improves on the ease-of-use of the end-user tools yet includes the power and flexibility of programming languages. This will require movement from both extremes — the end-user tools must incorporate more power, while the programming tools must incorporate greater ease-of-use. Perhaps eventually a product will emerge that allows people to converse conversationally with databases while performing their work.

This book addresses the gap that presently exists in ad hoc query tools by improving the ease-of-use of programming languages used for queries. For several years now, we've collected the questions and answers posted to online Oracle support groups about ad hoc queries. Principally, this involved monitoring the oracle-l listserv and, to a lesser extent, the Usenet group comp.databases.oracle. We also benefited from postings to the Oracle forum on CompuServe and to other online lists designed for specific products that use Oracle databases.

When printed and stacked, the assembled postings cast a formidable shadow. But even a cursory review of the postings showed that some questions get asked repeatedly or with slight variation. These frequently asked questions (or FAQs as they're called in the online lexicon) form the basis for this book.

We hope by having available to them a single resource that discusses the most frequently encountered ad hoc query problems, query writers will find it easier to access and use Oracle databases. There are, however, several important implications of our decision to focus on frequently asked questions.

- First, this book is not meant to be an exhaustive handbook of query techniques. The questions that get asked during online sessions clump around certain topics and leave other topics untouched. These gaps are reflected in the book. The categories of questions do prove interesting, however. For example, who could have predicted that among SELECT clauses the ORDER BY clause would generate the most problems?

- Second, the questions are somewhat time dependent, often relating to product cycles and enhancement schedules. For example, when Oracle version 7 first shipped, a flurry of questions appeared about the differences between CHAR and VARCHAR2. More recently, this question has disappeared as general familiarity with version 7 has improved. In this book we have restricted the questions to Oracle version 7. Specifically, the release versions we used were SQL*PLUS Release 3.2.2.0.1, PL/SQL Release 2.2.2.3.1, and Personal Oracle7 Release 7.2.2.3.1. The operating system was Windows 95.

- Third, many ad hoc query problems relate to a specific operating system and to specific hardware platforms. We have ignored all such questions and concentrated instead on questions concerning SQL, SQL*PLUS, and PL/SQL.

- Fourth, people who frequent online listservers have widely varying experience with Oracle products. Consequently, some questions reflect the efforts of people just learning ad hoc query techniques. Other questions pose problems stumbled on by people pushing a query product to its limitations. Most posted questions, however, fall somewhere between these two extremes. This book mirrors this distribution, with most questions falling somewhere in the middle of the degree of difficulty scale.

- Fifth, any book requires that organization be imposed on the topic. This organization, however, can have a profound effect on how easily a book serves as a query writing reference. We first organized the online postings in lumpy categories that evolved into chapters and then into subgroups that became section headings. However, this is only one organization among many possible ones, and it may not conform well with the way a particular query writer experiences a query problem. Consequently, we tried several techniques to make the material accessible. Each individual query problem concludes with a section entitled "See Also" that points to other

sections in the book dealing with related material. We also paid special attention to the index and included extensive cross-classifications and multiple headings for the same topic in an effort to make the index a really useful search tool.

We expect that shortly after print publication Komenda Publishing will provide a Web site devoted to the book. You will be able to browse the FAQs, search on keywords, download programs, email the authors, and review new material added after the book's publication.

Finally, a word about the conventions you'll find when reading this book. In the text we identify SQL, SQL*PLUS, and PL/SQL reserved words in SMALL CAPITALS. Whenever new concepts are first introduced, the defined words appear in *italics*. Many figures show the results of queries — which frequently occur in practice as computer printouts. To distinguish computer printouts, we've used the `OCRB monospaced font that looks like this`. All table names are shown in uppercase, bold, italic like ***ADDRESS***. All examples utilize simulated data, as opposed to any actual data that may have been posted to a listserv when a question was originally asked. Finally, one note of a grammatical nature. To avoid the awkward difficulty of maintaining gender neutrality (e.g., using "he or she"), we have simply alternated the use of masculine and feminine pronouns.

We've already been asked if there are really 101 questions in this book. The answer is no; there are actually 121 questions. We use "101" to mean a large number, but not yet large enough to qualify as 1001. Perhaps some day, as we add FAQs to the Web site, we may need to revise the book's subtitle.

This book is a modest effort, but we hope that query writers will find it a useful step toward the future of ad hoc queries that lies ahead.

Gary M. Lewis
Boulder, CO

Alex Sirota
Mississauga, Ontario