Limits of SQL Optimization

I work with SQL, which people spell S-Q-L and pronounce “sequel” like the sequel to Star Wars. Like other computer languages, SQL is just a set of characters you type in on the keyboard of a computer to tell it what to do. It has a set of grammatical rules just as English does but they are much simpler. SQL enables you to interact with a database. A database is a collection of information. Information means names, addresses, dollar amounts, dates, etc. SQL based relational database management systems, or RDBMS’s, take in statements written in the SQL language and use them to either read from or write to the database. Most web sites use SQL behind the scenes. For example, if you buy a book on Amazon.com the programs that run the Amazon web site use SQL to look up the book you want and to record your purchase. When an RDBMS gets a SQL statement, it chooses the fastest way to run it. Usually there are multiple ways to run a statement and the RDBMS has to choose the best option. Oracle calls the process of picking the best way to run a SQL statement “optimization” and they call the code that does it the “optimizer”. I have worked with the Oracle optimizer for over 19 years and I am still learning how it optimizes a SQL statement and why it sometimes makes mistakes. I am convinced that there are limits to how well any company’s SQL optimizer can work and I want to share that insight so that people who use SQL will have realistic expectations.

SQL saves time for web programmers by letting them say what to do on a database without saying how to do it. SQL stores information in tables. A table contains a collection of things of the same type. A table called CUSTOMERS could store your customers’ names and addresses. SQL relates tables together based on common attributes. You could associate a customer number with each customer. A table called SALES could record each item that you have sold. Each sale could have an associated customer number that allows you to relate sales and customers. However, a SQL statement that includes SALES and CUSTOMERS does not say which table to access first. The optimizer decides to access either sales first and then customers, or the other way around. Now consider how many possible orders there are in a SQL statement with ten tables. There are ten ways to pick the first table, nine ways to pick the second, etc. There are over 3 million possible orders to consider for a SQL statement with ten tables in it. In addition to the table order, the optimizer has to decide how to access each table. Each table can have multiple indexes. An index is just a fast way to find an entry in a table. You might have an index on customer number so you can quickly look up an entry in the CUSTOMERS table. The optimizer has to choose the best index for each table in a SQL statement. Now imagine that you have a SQL statement with ten tables and five indexes on each table. How many millions of ways are there that the optimizer could choose when it has to consider the three million orders for the tables and five different index possibilities for each table for each order considered? Without SQL a programmer would have to choose the order of the tables and which indexes to use. That is why SQL has taken over the web as the database language of choice. It frees web programmers from making all of these detailed decisions about how to access their information while allowing them to define what they want to do with the database.

You can override the Oracle optimizer’s choices of how to run a SQL statement if you can find a faster set of choices. You might choose a different table order or a different index. There have been many times where I have improved on the optimizer’s choices, often resulting in a SQL statement running a

thousand times faster with my choices than it did with the optimizer's choices. I have also learned different ways to configure a database so that the optimizer will make better choices on its own. Nevertheless, I keep running into statements where I have to override the optimizer's choices manually. I have a simple example of a two-table SQL query that I have not found any way to get Oracle's optimizer to run correctly. Every time I find a new performance feature, I try to see if it will run my example with the right choices, but nothing has. When the new 12c version of Oracle came out I tried each of the new performance features on my example and none of them caused the 12c optimizer to make the right choice. My intuition tells me that no matter what features Oracle or any other company comes out with they will not be able to make the right choices in all cases, but I cannot prove it. Still, experience shows that intervention is often required and anyone working with SQL should expect there to be statements that will not run efficiently unless a human being overrides the optimizer's wrong choices. It is a great tool, but like any tool, it has its limits.