Nicole Lawrence
Database Design & Development
CS225 Week3 Ponder Assignment

Report on Conditions, Expressions, and Operators

When accessing data in a database, one will seldom make a query without qualifying that search with conditions using the WHERE clause. This is because businesses almost always need data within a certain time frame, with a certain ID, or from a certain location. To narrow SQL queries accordingly, the user adds WHERE to the SELECT, UPDATE, and DELETE clauses. This signals the server to look for conditions limiting the data retrieved. There is also the inverse, WHERE NOT, which is the same as saying 'does not equal to'.

Before continuing, it is important to note the little niceties that make a query more user-friendly. A single condition within the WHERE clause is intelligible. Multiple conditions stacked on top of each other, while understandable by the server, is more difficult for the user to quickly read. Adding parentheses to each separate condition mitigates this difficulty. Similarly, using WHERE NOT instead of the more server-friendly <> operator is a stylistic decision that can be made.

The conditions mentioned in the above paragraph are made up of one or more expressions combined with one or more operators. An expression could be a number, column in a table or view, string literal, built-in function, subquery, or list of expressions. The available operators are those familiar to other programming languages: =, !=, <, >, <>, +, -, *, /. As well, there is the addition of BETWEEN, IN, and LIKE. In short, BETWEEN returns a specified range or time frame without having to separate conditions. This encompasses numeric and string literals ranges, and is inclusive (the upper and lower limits are included in any results). A minor but important note: the lower limit of the range must precede the upper limit in the condition. When grouping together OR expressions, it is useful to utilize the IN operator. The syntax for this is shown in this example: SELECT title, rating FROM film WHERE rating IN ('G', 'PG');. This is a handy way to improve readability while creating an effective query. There is an inverse for in: NOT IN which obviously returns the opposite of what IN would.

The final operator, LIKE, can be used to compare partial strings with possible database results. Subqueries with LIKE == '%string%' return data screened to only include matches with that specific string. This can also be accomplished with LIKE == 'REGEX'. Regular Expressions are commonly used in programming and any self-respecting programmer should take the time to learn it. The NULL expression should also be familiar. Queries, especially those in a new database, should put in safeguards to avoid or account for NULLs.