Restricting Rows

WHERE Clause

The WHERE clause is an option on the SELECT statement that allows certain rows to be selected from the table. The basic use is: WHERE column name = value. I like to use the analogy that the WHERE condition is like an IF statement in any other computer language. "If this column in this row has this value, then select that row into the results."

In general, the same rules of logic apply when comparing a column to another column or constant:

- The column and the value it's being compared to should be of the same data type
- Conditions can be combined to form logical AND/OR/NOT situations and are evaluated using logical truth values.
- The order of operations is arithmetic operators, then comparison operators, then logical operators. Parenthesis can be used to affect the order of operations.
- Mathematical comparison operators (=, <, >, <=, >=, <>, !=, ^=) are very similar to other languages in their usage.

When comparing a text type column to a string literal, single quotes should be used. Double quotes are used only for column aliases. Also bear in mind that the string literal should be case sensitive to match how the data is actually stored in the table. Date columns should compare to a string literal in the format of 'DD-MON-YY', so June 17, 2010 would be '17-JUN-10'. When comparing a numeric column to a numeric literal, no quotes should be used for proper comparison. For queries with a calculation in the SELECT clause, the actual calculation must be used in the WHERE clause and not the column alias.

The WHERE clause has other operators that can be quite useful. The BETWEEN operator allows a range of values to be specified, in other words, a lower and upper limit to the condition, such as WHERE amount BETWEEN 10 AND 20. The equivalent to this without the BETWEEN operator would be WHERE amount >= 10 AND amount <= 20. Note that the value 10 and the value 20 would meet the condition.

The IN operator allows a value to be searched in a "subset" of values. This can save quite a bit of coding, depending on the number of values being searched. As an example, we might want rows that have a zip code of 52501, 52537 or 52509. Without the IN operator, it would look like this: WHERE zip = '52501' OR zip = '52537' OR zip = '52509'. Notice that each comparison is a complete logical expression. With the IN operator, it shortens to: WHERE zip IN ('52501', '52537', '52509'). As you can see, the more values specified, the more useful it becomes. It can even be used on a single value: WHERE zip IN ('52501'), although unlikely. Notice the list is enclosed in parenthesis and each value is separated by a comma. These examples assume zip code is a text type column.

The LIKE operator allows searching to be done using pattern matching criteria. For instance, if you wanted to shows all rows that have a zip code starting with 525, then we would code: WHERE zip LIKE

'525%'. The string literal starts with 525, then uses the percent sign (%) wildcard character to tell SQL that any number of characters after the 525 match our criteria. So zip codes 52501 and 52537-4011 would be selected, but not 52431, for example. The other wildcard character is the underscore (_) which means a single character. Back to our example, if we wanted only zip codes that have a 5 in the first position and a 5 in the third position, it would be coded as: WHERE zip LIKE '5_5%'.

NULL values are a little tricky in SQL. Recall that a NULL means absent of value. To check a column to see if it contains a NULL value, you must use the IS NULL comparison operator, and not the equal (=) operator.

ORDER BY Clause

Most generally the results from a query need to be sorted in a different order than how they are stored on the table. To sort data in SQL, we use the ORDER BY clause. The ORDER BY clause is optional and if used must be coded after the WHERE clause (if there is one), or after the FROM clause if there is no WHERE clause. This is because it is more efficient to "filter" the records first, and then sort them, as opposed to sorting the entire table then filtering them. To sort by zip code in ascending order, we would code: ORDER BY zip

The default sort order is ascending, but can be changed to descending order by using the DESC operator. Ascending order is as follows: blank/special characters, numeric values, character values (uppercase then lowercase), NULL values. To order by zip code in descending order, we would code: ORDER BY zip DESC. NULL values can be sorted to the top by coding NULLS FIRST after the column name.

More than one column can be coded on the ORDER BY. The first column would be the primary, and the second column listed would be the secondary. This allows for a "sort by this field (secondary) within this field (primary)" scenario. If we wanted to sort by zip code (secondary) within state (primary), we would code: ORDER BY state, zip. The results of this query would show the records grouped (sorted) by state, then within each set of state records, records would be grouped (sorted) by zip code. The number of columns that can be coded on the ORDER BY is limited to 255. You can also use the column reference number in the ORDER BY clause instead of the column name in the SELECT.

Take look at the following queries, which both give the same result:

```
SELECT name, zip, salary * 2
FROM Customer
ORDER BY salary * 2;

SELECT name, zip, salary * 2
FROM Customer ORDER
BY 3;
```