Calculations in SQL

Class Notes

To have a 3NL database, you should not have any fields that can be calculated from other fields - that's just a waste of space. Presumably, then, there must be some way of calculating new data from existing fields.

Formulae

To add mathematical formula to a query is pretty much what you would expect. Say you had a "Traffic" table with, among other things, a "speed" field which stores readings from a speed camera. Now, say you found out the camera was calibrated incorrectly and read everyone's speed as 5% higher than it should be. Rather than throw away the data, you can adjust it on the fly during a query. So, to reduce display the speed reduced by 5%, we can use…

```
SELECT speed * 0.95 FROM Traffic;
```

It's that simple. We can use all the usual mathematical symbols, brackets and so on. So, if you wanted to do it the long way, you could do this…

```
SELECT speed - (speed / 100 * 5) FROM Traffic;
```

You can also use multiple fields in the one equation. Say we wanted to find out how much faster a car could go than the speed they were clocked at.

```
SELECT maximumSpeed - speed FROM Traffic.
```

You can also give the calculated field a name, which will be displayed in any situation where normal field names would be displayed. So, for example, on the first row of a report.

```
SELECT maximumSpeed - speed as "Extra Speed Available" FROM Traffic;
```

Unless you're just doing something very quickly just for yourself, you should always name your calculated fields.

Functions

SQL also provides spreadsheet-like functions for common tasks. For example, if you wanted the average of the speed all the cars were going using the table in the previous example, you would use…

```
SELECT AVG(speed) FROM Traffic;
```

There is also COUNT() that returns how many of something there is (often used in conjunction with a WHERE clause, so if you wanted to know how many people were speeding, you could use…

```
SELECT COUNT(speed) FROM Traffic WHERE speed > 60;
```

And there is also SUM(), which will give you a total.

Finally, you can combine functions. So, if you wanted to calculate your own average for some reason, you could use…

```
SELECT SUM(speed) / COUNT(speed) FROM Traffic;
```

Both At Once

Using both calculations and functions at the same time is extra complicated, though. Say you wanted to know which cars were going at below average speed. Logically, you could do something like this…

```
SELECT numberPlate FROM Traffic WHERE speed < AVG(speed);
```

However, this will cause an error.

The problem is that we're asking for two different types of queries. If you use a query with a function call, you will get back just one piece of data - an average or a total, for example. If you use a query with a calculated field, you will get back a list of records. This query is asking for both, and even though it makes perfect sense to use reading it, SQL needs it split into two queries.

So, could we do one query to get the average and maybe store it in a variable or something, then use it in the second query? No. SQL is not a programming language and doesn't have variables.

The solution is to use a nested query - a query inside a query.

```
SELECT numberPlate FROM Traffic WHERE speed < (SELECT AVG(speed) FROM Traffic);
```

This may seem annoyingly complicated now but if you think about it, it's also very powerful. The nested queries don't have to refer to the same table, after all. With nested queries, we can do calculations using data from multiple tables.