QL technique: functions

metimes, the information that we need is not actually stored in the tabase, but has to be computed in some way from the stored data. In r order entry example, there are two derived attributes (/subtotal in rderLines and /total in Orders) that are part of the class diagram but not part of the relation scheme. We can compute these by using SQL functions in the SELECT statement.

There are many, many functions in any implementation of SQL—far more than we can show here. Unfortunately, many of the functions are defined quite differently in different database packages, so you should always consult a reference manual for your specific software.

Computed columns

We can compute values from information that is in a table simply by showing the computation in the SELECT clause. Each computation creates a new column in the output table, just as if it were a named attribute.

Example: We want to find the subtotal for each line of the OrderLines table, just as shown in the UML class diagram. Obviously, the total of each line is simply the unit sale price times the quantity ordered, so we don't even need a function yet—just the computation. We have included all three of the OrderLines PK attributes in the SELECT clause attribute list, to be sure that we show the subtotal for each distinct line.

```
SELECT custID, orderDate, UPC, unitSalePrice * quantity
FROM orderlines;
```

Order line subtotals

5678	2003-07-14	51820 33622	11.95
9012	2003-07-14	51820 33622	23.90
9012	2003-07-14	11373 24793	21.25
5678	2003-07-18	81809 73555	18.00
5678	2003-07-20	51820 33622	23.90
5678	2003-07-20	81809 73555	9.00
5678	2003-07-20	81810 63591	24.75

Computations are not limited just to column names; they may also include constants. For example, unitsaleprice * 1.06 might be used to find the sale price plus sales tax.

Notice that the computation itself is shown as the heading for the computed column. This is awkward to read, and doesn't really tell us what the column means. We can create our own column heading or alias using the **AS** keyword as shown below. (In fact, we could simply write the new name of the column without saying AS. Please don't do this—it hurts readability of your code.) If you want your column alias to have spaces in it, you will have to enclose it in *double* quote marks.

```
SELECT custID, orderDate, UPC,
  unitSalePrice * quantity AS subtotal
FROM orderlines;
```

Order line subtotals

5678	2003-07-14	51820 33622	11.95
9012	2003-07-14	51820 33622	23.90
9012	2003-07-14	11373 24793	21.25
5678	2003-07-18	81809 73555	18.00
5678	2003-07-20	51820 33622	23.90
5678	2003-07-20	81809 73555	9.00
5678	2003-07-20	81810 63591	24.75

Aggregate functions

SQL **aggregate functions** let us compute values based on multiple rows in our tables. They are also used as part of the SELECT clause, and also create new columns in the output.

Example: First, let's just find the total amount of all our sales. To compute this, all we need is to do is to add up all of the price-times-quantity computations from every line of the OrderLines. We will use the **SUM** function to do the calculation. The output table, as you should expect, will contain only one row.

```
SELECT SUM(unitSalePrice * quantity) AS totalsales
FROM orderlines;
```

Sales

132.75

Next, we'll compute the total for each order (the derived attribute shown in the UML Order class). We still need to add up order lines, but we need to group the totals for each order. We can do this with the **GROUP BY** clause. This time, the output will contain one row for every order, since the customerID and orderDate form the PK for *Orders*, not OrderLines. Notice that the SELECT clause and the GROUP BY clause contain exactly the same list of attributes, except for the calculation. In most cases, you will get an error message if you forget to do this.

```
SELECT custID, orderDate, SUM(unitSalePrice * quantity) A
FROM orderlines
GROUP BY custID, orderDate;
```

Order totals

5678	2003-07-14	11.95
9012	2003-07-14	45.15
5678	2003-07-18	18.00
5678	2003-07-20	57.65

Other frequently-used functions that work the same way as SUM include MIN (minimum value of those in the grouping), MAX (maximum value of those in the grouping, and AVG (average value of those in the grouping).

The COUNT function is slightly different, since it returns the *number* of rows in a grouping. To count all rows, we can use the * (for example, to find out how many orders were placed).

```
SELECT COUNT(*)
FROM orders;
```

Orders

4

We can also count groups of rows with identical values in a column. In this case, COUNT will ignore NULL values in the column. Here, we'll find out how many times each product has been ordered.

```
SELECT prodname AS "product name",
   COUNT(prodname) AS "times ordered"
FROM products NATURAL JOIN orderlines
GROUP BY prodname;
```

Product orders

Hammer, framing, 20 oz.	3
Saw, crosscut, 10 tpi	1
Screwdriver, Phillips #2, 6 inch	2
Pliers, needle-nose, 4 inch	1

A WHERE clause can be used as usual before the GROUP BY, to eliminate rows before the group function is executed. However, if we want to select output rows based on the results of the group function, the **HAVING** clause is used instead. For example, we could ask for only those products that have been sold more than once:

```
SELECT prodname AS "product name",
   COUNT(prodname) AS "times ordered"
```

```
        FROM products NATURAL JOIN orderlines
        GROUP BY prodname
        HAVING COUNT(prodname) > 1;
```

Other functions

Most database systems offer a wide variety of functions that deal with formatting and other miscellaneous tasks. These functions tend to be proprietary, differing widely from system to system in both availability and syntax. Most are used in the SELECT clause, although some might appear in a WHERE clause expression or an INSERT or UPDATE statement. Typical functions include:

- Functions for rounding, truncating, converting, and formatting numeric data types.
- Functions for concatenating, altering case, and manipulating character data types.
- Functions for formatting dates and times, or retrieving the date and time from the operating system.
- Functions for converting data types such as date or numeric to character string, and vice-versa.
- Functions for supplying visible values to null attributes, allowing conditional output, and other miscellaneous tasks.