# Chapter 1-

## Objectives

The primary objective of this lab is to gain familiarisation with the subqueries in SQL.

## Reference Material

This practical is based on material in the chapter.

## Overview

In this exercise you will use the QAStore database to investigate and write a number of SQL subqueries.

## Estimated duration

The estimated duration for this lab is 30 minutes.

## Completed solution

A Visual Studio 2010 solution containing the fully completed code for this lab is located in *CoursewareFolder*\1 \Solutions.

## Step by Step

1. Start SQL Server Management studio from the Windows Start button.
2. Enter .\SQLEXPRESS as the Server name in the Connect to Server dialog box.
3. Choose QAStore from the drop down list of available databases located on the standard toolbar to ensure that it is selected as the current database.
4. Click the New Query button on the standard toolbar.

### Part 1 (Standard Subqueries)

### Exercise 1

Show the full name of the salesperson(s) (could be more than one) who has/have the largest sales target.

If you do it right then the results should look like the following (1 row):

fname lname

--------------- ---------------

Billy Custard

**Question:** Did you solution use ‘=’ or ‘IN’ would it be safe to use either? Why?

### Exercise 2

We would like to write a query that displays the number '3'.

Why? Because that is the answer to "How many people have sold?".

Before starting, run the two queries supplied below, and work out the result (3) manually. We think there are only 2 ways of doing it. See if you can work out the 2 ways.

SELECT emp\_no AS 'emp nos of the people' FROM salesperson

SELECT emp\_no AS 'emp nos who have sold' FROM sale ORDER BY emp\_no

Now try to write SQL that mimics exactly what you just did manually. It can be done using a simple subquery and/or you may think of a way of doing it without a subquery. Either way don't try any sort of JOIN!

The answer is 3 so surely either coded solution query must start with:

SELECT COUNT(...

All you need to do is complete the rest!

### Part 2 (Slightly harder subqueries and WHERE EXISTS)

### Exercise 3

Display firstname, lastname, sales\_target and the total of the sales(order\_value) each salesperson has achieved.

We should ensure every salesperson is in the report not just the 3 who have sold. There should be 1 row in the output for each salesperson (so 6 rows & 4 columns).

You know how to do this already, nothing new here, so to save time we are going to build up the query together

The SELECT list is surely:

SELECT fname, lname, sales\_target, SUM(order\_value) 'Sales achieved'

It clearly cannot run without a GROUP BY, the SELECT clause determines the GROUP BY:

SELECT fname, lname, sales\_target, SUM(order\_value) 'Sales achieved'

GROUP BY fname, lname, sales\_target

What's missing? A FROM clause. We need 2 tables joined and we want all salespeople, not just the 3 who have sold so we must do an Outer join.

We’ve already covered Joins & Outer Joins, so complete the following code fragment by keeping/removing the word 'LEFT' or the word 'RIGHT' in the appropriate place and write the correct ON clause.

SELECT fname, lname, sales\_target,

sum(order\_value) 'Sales achieved'

FROM salesperson SP LEFT/RIGHT JOIN sale S

-- type 'LEFT' or 'RIGHT' you choose!

ON ?? = ??

GROUP BY fname, lname, sales\_target

Ensure you now have 6 rows 4 columns, note the NULL's in the last column

fname lname sales Sales

target achieved

---------- ---------- -------- --------------

Alan Brick 9.00 8

Billy Custard 14.00 NULL

Chris Digger 7.00 NULL

Dick Ernst 11.00 NULL

Ernest Flipper 12.00 27

Fred Goalie 13.00 30

### Exercise 4

Take the solution to exercise 3 and use the COALESCE function to ensure that the total sales of the 3 people who have not sold is 0 and not NULL.

**Hint**: pass the result of the SUM function as one of the arguments to the COALESCE function.

If you do it right then the results should look like the following (6 rows, 4 columns):

fname lname sales Sales

target achieved

---------- ---------- -------- --------------

Alan Brick 9.00 8

Billy Custard 14.00 0

Chris Digger 7.00 0

Dick Ernst 11.00 0

Ernest Flipper 12.00 27

Fred Goalie 13.00 30

### Exercise 5

Let's now assess who would be interested in seeing this output.

* The sales manager / sales director?

What would he/she be MOST interested in?

* They would definitely be interested in how the sales\_targets compare to the sales achieved for each individual so that bonuses / warnings could be issued as appropriate,
* But even more interested in "does the total sales achieved by all 6 exceed the total of the sales\_targets of all 6?" - Because that is probably what the sales manager's bonus is based on!!

You have been taught that WHERE EXISTS is used when you want to make an outer query run or not run based on whether a subquery produces any output. We know that the sum of the targets of the 6 people is 66. We also know that the sum of the order values of the 8 sales is currently 65.

So the following query would run. Check it.

SELECT fname, lname, sales\_target, COALESCE(SUM(order\_value),0) 'Sales achieved'

FROM salesperson SP LEFT JOIN sale S

ON SP.emp\_no = S.emp\_no

WHERE EXISTS

(

SELECT 'gobbledegook'

WHERE 66 > 65

)

GROUP BY fname, lname, sales\_target

Change the 65 to 67 (mimicking an extra sale of 2) and it won't run (check it). Obviously what we have to do is get rid of the hard coded numbers 65 & 66 and write SQL that calculates the correct ongoing values, so that the report is only produced if the following statement is true:

"The sum of the sales targets of all the sales people combined is greater than the total value of all the orders in sale".

I.E. The salespeople as a WHOLE have not hit their COMBINED target.

This query is NOT based on any individual or dept but on the sum of everyone!

So, you might think that this nested query solves the problem

SELECT fname, lname, sales\_target, COALESCE(SUM(order\_value),0) 'Sales achieved'

FROM salesperson SP LEFT JOIN sale S

ON SP.emp\_no = S.emp\_no

WHERE EXISTS

(

SELECT 'Whatever'

FROM salesperson SP1 JOIN sale S1

ON SP1.emp\_no = S1.emp\_no

HAVING SUM(sales\_target) > SUM(order\_value)

)

GROUP BY fname, lname, sales\_target

However can you spot the 'bug'?

The code is INCORRECT because it double/triple.counts sales targets in the subquery

This can be clearly seen if you run this query

SELECT SUM(sales\_target) as 'Total of Targets' , SUM(order\_value) as 'Total of Sales'

FROM salesperson SP1 JOIN sale S1

ON SP1.emp\_no = S1.emp\_no

HAVING SUM(sales\_target) > SUM(order\_value)

**Explanation:**

Alan Brick (emp\_no 10) has a sales target of 9 and has two entries in the sales table (of 5 and 3). Using the above query SUM(sales\_target) would therefore give 2 \* 9 = 18. SUM(order\_value) would give (5 + 3) = 8

Ernest Flipper (emp\_no 50) has a sales target of 12 and has one entry in the sales table (of 27). Using the above query SUM(sales\_target) would therefore give 1 \* 12 = 12. SUM(order\_value) would give 27

Fred Goalie (emp\_no 60) has a sales target of 13 and has five entries in the sales table (of 7, 6, 12, 2 and 3). Using the above query SUM(sales\_target) would therefore give 5 \* 13 = 65. SUM(order\_value) would give (7 + 6 + 12 + 2 + 3) = 30

All the others have no entries in the sales table. Using the above query SUM(sales\_target) would therefore give 0 \* sales\_target = 0. SUM(order\_value) would give 0.

Adding up the sales\_targets gives 18 + 12 + 65 = 95 (the result you get when you run the above query) which is plainly meaningless and wrong. It should be 66 => (9 + 14 + 7 + 11 + 12 + 13).

Adding up the order\_values gives 8 + 27 + 30 = 65.

In both situations (the wrong and correct calculations of SUM(sales\_target) are greater than SUM(order\_value) so the query produces a result.

**MORAL**

**NEVER JOIN 1 to MANY AND USE A SUM FROM THE ONE!!!**

The correct solution is:

SELECT fname, lname, sales\_target, COALESCE(SUM(order\_value),0) 'Sales achieved'

FROM salesperson SP LEFT JOIN sale S

ON SP.emp\_no = S.emp\_no

WHERE EXISTS

(

-- after running the inner code

-- sum(sales\_target) will contain some data,

-- so the EXISTS above will be true and the

-- Outer Query will run

SELECT sum(sales\_target)

FROM salesperson

-- total of sales\_targets is produced second

HAVING sum(sales\_target) >

(

-- this nested query runs first

SELECT sum(order\_value)

FROM sale

)

)

GROUP BY fname, lname, sales\_target

It sums the order\_values for each sale and does this once for each sale regardless of who made the sale. It then compares this total with the sum of the targets of each salesperson (regardless of whether they have sold or not) and is how you would do it manually.

### Exercise 6

Now that we have the solution to exercise 5 working, let’s decide that if the report is produced then the person who sees it only needs to see the rows representing the salespeople who have NOT hit their targets (4 of the 6).

Make a small change to your solution to exercise 5 by adding a HAVING clause to the bottom of the query that reduces the answer set to produce just 4 rows (as only 2 sales people have hit their target).

**Note:** You do not need to change any code you have written, just add some more!!

Remember there is nothing that is equal to NULL, so in a similar fashion nothing is ever 'greater than' or 'less than' NULL!!

SELECT fname, lname, sales\_target,

COALESCE(SUM(order\_value),0) 'Sales achieved'

FROM salesperson SP LEFT JOIN sale S

ON SP.emp\_no = S.emp\_no

WHERE EXISTS

(

SELECT SUM(sales\_target)

FROM salesperson

HAVING SUM(sales\_target) >

(

SELECT SUM(order\_value)

FROM sale

)

)

GROUP BY fname, lname, sales\_target

-- complete the following line

HAVING ?????? > ???????????

If you do it right then the results should look like the following (4 rows, 4 columns):

fname lname sales Sales

target achieved

---------- ---------- -------- -------

Alan Brick 9.00 8

Billy Custard 14.00 0

Chris Digger 7.00 0

Dick Ernst 11.00 0

**Exercise 7**

The following line from the result set catches your eye

fname lname sales Sales

target achieved

---------- ---------- -------- -------

Alan Brick 9.00 8

What business decision can you sensibly make given that you know what the underlying SQL is doing?

Is it safe to assume Alan is just under target and will exceed it with the next sale?

What you must consider is this: If there are 2 salespeople both called Alan Brick, will they get:

1. A row each in the output

or

1. Be put in the same bucket, giving us one row in the output representing all people called 'Alan Brick'?

Please decide now. Make a decision. Don't worry if it is wrong.

When you’ve made your decision look on the next page to see the answer.

The answer is neither!!

If two salespeople called Alan Brick had the SAME sales\_target then they would be treated as 1 person and there would be one 'Alan Brick' row showing the result of their combined performance

Remember the grouping is by fname, lname, & SALES\_TARGET, not just by name(s).

The problem is that the query does not contain emp\_no. You might say it does, but it doesn’t. It's appearance in the ON clause to make the correct JOIN is a complete irrelevance with regard to the GROUP BY.

So making the following change to the GROUP BY of the query ensures that each 'Alan Brick' gets his own row, even if they have the same sales\_target.

SELECT fname, lname, sales\_target,

COALESCE(SUM(order\_value),0) 'Sales achieved'

FROM salesperson SP LEFT JOIN sale S

ON SP.emp\_no = S.emp\_no

WHERE EXISTS

(

SELECT SUM(sales\_target)

FROM salesperson

HAVING SUM(sales\_target) >

(

SELECT SUM(order\_value)

FROM sale

)

)

GROUP BY SP.emp\_no, fname, lname, sales\_target

--------\* - new column added

HAVING sales\_target > COALESCE(SUM(order\_value),0)

The problem then is that potentially in the future the output might look like this:

fname lname sales\_target Sales achieved

---------- --------- -------------- --------------

Alan Brick 9.00 6

Alan Brick 9.00 2

Billy Custard 14.00 0

Chris Digger 7.00 0

Dick Ernst 11.00 0

but with no guarantee that the two 'Alan Brick' rows are anywhere near each other or even on the same page of the report.

**Question**: How would your 'users' read the above query? Would they realize that the 2 rows represent 2 different people?

Surely if SP.emp\_no is in the GROUP BY, it should be in the SELECT list,

i.e:

SELECT SP.emp\_no, fname, lname, sales\_target,

--------\* -- column added

COALESCE(SUM(order\_value),0) 'Sales achieved'

FROM salesperson SP LEFT JOIN sale S

ON SP.emp\_no = S.emp\_no

WHERE EXISTS

(

SELECT SUM(sales\_target)

FROM salesperson

HAVING SUM(sales\_target) >

(

SELECT SUM(order\_value)

FROM sale

)

)

GROUP BY SP.emp\_no, fname, lname, sales\_target

HAVING sales\_target > COALESCE(SUM(order\_value),0)

So the output would now be (with one Alan Brick)

emp\_no fname lname sales Sales

target achieved

------- --------- ------- --------- --------

10 Alan Brick 9.00 8

20 Billy Custard 14.00 0

30 Chris Digger 7.00 0

40 Dick Ernst 11.00 0

and, if there were 2 Alan Bricks, then the output would show clearly that they are 2 different people.

emp\_no fname lname sales Sales

target achieved

------- --------- ------- --------- --------

10 Alan Brick 9.00 8

20 Billy Custard 14.00 0

30 Chris Digger 7.00 0

40 Dick Ernst 11.00 0

90 Alan Brick 9.00 2

**THE MORAL IS:**

"WRITE THE CORRECT 'GROUP BY' FIRST & THEN COPY/PASTE INTO THE 'SELECT' LIST"

NOT THE REVERSE as you might have surmised from earlier study/lectures.

### Exercise 8

Can you achieve the same output as exercise 7 WITHOUT using EXISTS? You should only have to make a small change in your code.

**Hint:** make the WHERE clause look like:

WHERE (subquery returning one number) > (subquery returning another number)

**WARNING:** The solution is on the next page

SELECT SP.emp\_no, fname, lname, sales\_target,

COALESCE(SUM(order\_value),0) 'Sales achieved'

FROM salesperson SP LEFT JOIN sale S

ON SP.emp\_no = S.emp\_no

WHERE

(SELECT sum(sales\_target) FROM salesperson)

>

(SELECT sum(order\_value) FROM sale)

GROUP BY SP.emp\_no, fname, lname, sales\_target

HAVING sales\_target > COALESCE(SUM(order\_value),0)

Everyone would agree that this code is:

1. simpler to read
2. simpler to write
3. easier to maintain
4. only works because these subqueries each return a 1 row 1 column 'atomic' number.

We have NOT broken the rule of "cannot use an aggregate in a WHERE clause"; we used it in a subquery of the WHERE clause, an important distinction.

In SQL, as in life, there are many ways of achieving the same end!

As a student your question might be "which will be faster?"

The answer is there are too many factors to determine this easily, but if your DBMS has a 'show execution plan/costs' feature you could run the two together and see.

Last time we ran the code of exercise 7 and 8 alongside each other the Execution plans in SQL SERVER 2008 were similar and the cost of the query using EXISTS was 51% of the batch whilst the query using the WHERE clauses cost 49% of the batch which suggests there’s very little in it.

### Exercise 9

How do we find the names associated with the top 2 sales\_target(s)?

Perhaps the following would do the trick?

SELECT fname, lname, sales\_target

FROM salesperson

ORDER BY sales\_target DESC

fname lname sales\_target

--------------- --------------- --------------

Billy Custard 14.00

Fred Goalie 13.00

Ernest Flipper 12.00

Dick Ernst 11.00

Alan Brick 9.00

Chris Digger 7.00

That does it. Just read the top 2 rows from the output of 6 rows.

However, this is somewhat inefficient if you had 600/6000/60000 rows.

Here’s a nice SQL Server solution:

SELECT TOP 2 fname, lname, sales\_target

FROM salesperson

ORDER BY sales\_target DESC

fname lname sales\_target

--------------- --------------- --------------

Billy Custard 14.00

Fred Goalie 13.00

But what if there was another person who also had a target of 13, could I get them as well?

Yes, via 'with ties'.

SELECT TOP 2 WITH TIES fname, lname, sales\_target

FROM salesperson

ORDER BY sales\_target DESC

-- You can base your question on percentages:

SELECT TOP 35 PERCENT WITH TIES fname, lname, sales\_target

FROM salesperson

ORDER BY sales\_target DESC