# Chapter 1- Correlated Subqueries

## 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 *Correlated Subqueries* 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 Correlated Subqueries\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.

### Correlated Subqueries and how to work around them

Correlated subqueries are quite hard. You will occasionally need them, but they can often be coded in another way that you might consider easier, or maybe harder, - we will show you both.

**Tutorial Portion**. We would like you to step thru the following steps at your own speed:

1. An introduction to a new sort of 'problem'
2. A first stab might be
3. Getting rid of the MAX/GROUP BY
4. Enhancing the simplistic subquery
5. The fatal flaw, and proof
6. Moving towards the solution
7. Mandatory aliasing needed and solution achieved
8. Tidying up the solution
9. Correlated queries are all the same
10. The ALTERNATIVE - a subquery in the FROM clause, 'an inline VIEW'!!

### 1 An introduction to a new sort of 'problem'

Imagine we want to find the biggest sale per company. It sounds easy. Lets list the sales sorted by order\_value within company\_no first, so that we can easily work it out manually and then try and build up some code.

SELECT company\_no, order\_value

FROM sale

ORDER BY company\_no, order\_value

company\_no order\_value

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

1000 5 --no

1000 7 --yes, biggest for company 1000

2000 3 --no

2000 12 --yes, biggest for company 2000

3000 3 --no

3000 6 --no

3000 27 --yes, .......3000

4000 2 --yes, .......4000

So the 2nd, 4th, 7th and 8th rows are the biggest per company.

The answer set we are trying to produce (via code) is therefore:

company\_no order\_value

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

1000 7

2000 12

3000 27

4000 2

Seems easy enough, let's try.

### 2 A first stab might be

SELECT company\_no, MAX(order\_value) AS 'Biggest sale per company'

FROM sale

GROUP BY company\_no

Should produce :

company\_no Biggest sale per company

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

1000 7

2000 12

3000 27

4000 2

Problem solved, or a very minimalist answer?

We could easily JOIN to 'company' to pick up 'name' of company.

But what if we want to display some additional things like: 'date' of sale, 'emp\_no' who made sale, 'description' .etc.?

Let's just add 'emp\_no' to the SELECT list of last query and see what happens:

SELECT company\_no, emp\_no, MAX(order\_value) AS 'Biggest sale per company'

FROM sale

GROUP BY company\_no

Unfortunately when you try to run this you get the following error message:

"Column 'sale.emp\_no' is invalid in the SELECT list because it is not contained in either an aggregate function or the GROUP BY clause."

We have broken a golden rule - "you must GROUP BY everything you SELECT that is not being aggregated". Basically, 'emp\_no' if 'SELECT'ed must be in Group by, so let’s try the following:

SELECT company\_no, emp\_no, MAX(order\_value) AS 'Biggest sale per company per employee'

FROM sale

GROUP BY company\_no, emp\_no

This should give:

company emp Biggest sale per

no no company per employee

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

1000 10 5

2000 10 3

3000 50 27

1000 60 7

2000 60 12

3000 60 6

4000 60 2

The altered column heading of the last column gives the game away. We have "Changed the Question by changing the GROUP BY" clause.

Our problem is really caused by the MAX in the SELECT list forcing us to have a 'GROUP BY' clause.

### 3 Getting rid of the MAX/GROUP BY

What we need is a query that does

SELECT \*

and **no** 'GROUP BY' clause, but with a 'WHERE' clause that manages to find just 4 of the 8 rows

So let's start.

SELECT \*

FROM sale

WHERE order\_value =

We know this is a good start because PRIMARILY the row is being SELECTed because of the contents of the 'order\_value' column and not anything else.

The right hand side of the 'WHERE' clause must be some calculated 'MAX', so a subquery is needed.

SELECT \*

FROM sale

WHERE order\_value =

(

SELECT MAX(order\_value)

FROM sale

)

ORDER BY company\_no

This runs, but produces the wrong answer:

ord emp their cmpy cntct ord order desc

no no ord no no code val date

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

600 50 AC12 3000 PP 27 2006-05-23... Complete Desk...

The reason being the subquery produces a one number answer of 27. So we just get sales whose value is 27, i.e. the biggest of the whole population of 8 sales.

### 4 Enhancing the simplistic subquery

We actually want (let’s just remind you of the original list):

cmpy order

no value

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

1000 5

-- this row selected because it's value = biggest of rows 1 & 2

1000 7

2000 3

-- this row selected because it's value = biggest of rows 3 & 4

2000 12

3000 3

3000 6

-- this row selected because it's value = biggest of rows 5, 6 & 7

3000 27

-- this row selected because it's value = biggest of row 8 only

4000 2

So let's make the subquery come back with 4 numbers namely 7, 12, 27, and 2. These are the 4 maximums. This is easily done by adding a 'GROUP BY' to the SUBQUERY:

SELECT \*

FROM sale

WHERE order\_value =

(

SELECT MAX(order\_value)

FROM sale

-- this line added

GROUP BY company\_no

)

ORDER BY company\_no

-- Let's just run the subquery alone.

SELECT MAX(order\_value) AS biggest

FROM sale

GROUP BY company\_no

The subquery produces:

biggest

-----------

7

12

27

2

If we now run the whole query we get the following error message:

"Subquery returned more than 1 value. This is not permitted when the subquery follows =, !=, <, <= , >, >= or when the subquery is used as an expression".

This is pretty self explanatory, so let’s change the comparison operator '=' to an 'IN'.

SELECT \*

FROM sale

-- line altered from '=' to 'IN'

WHERE order\_value IN

(

SELECT MAX(order\_value)

FROM sale

-- this line added

GROUP BY company\_no

)

ORDER BY company\_no

This should produce:

ord emp their cmpy cntct ord order desc

no no ord no no code val date

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

100 60 AA1 1000 MM 7 2006-06-24... Toshiba 6700 Pro

300 60 Ord39 2000 OO 12 2006-07-14... ScanPRO 4800 S...

600 50 AC12 3000 PP 27 2006-05-23... Complete Deskt...

500 60 DPF78 4000 TT 2 2006-07-23... Laser printer

So it appears to produce the right result but is there a fatal flaw in the logic?

### 5 The fatal flaw, and proof

There’s a fatal flaw I’m afraid! There’s a built in assumption that order\_values are unique. It assumes wrongly that a sale whose order\_value happens to be the biggest for its company, could not at the same time be equal to the 2nd biggest for another company or even the 17th biggest to yet another company.

Temporarily INSERT this row:

A sale whose value is 12 to company 3000, and is equal to the biggest to a different company (company 2000).

INSERT INTO sale VALUES(900, 60, '001' 3000, 'PP', 12, NULL, 'someitem')

(1 row(s) affected)

Now rerun current version.

SELECT \*

FROM sale

-- line altered from '=' to 'IN'

WHERE order\_value IN

(

SELECT MAX(order\_value)

FROM sale

-- this line added

GROUP BY company\_no

)

ORDER BY company\_no

It will give you

ord emp their cmpy cntct ord order desc

no no ord no no code val date

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

100 60 AA1 1000 MM 7 2006-06-24... Toshiba 6700 Pro

300 60 Ord39 2000 OO 12 2006-07-14... ScanPRO 4800 S...

600 50 AC12 3000 PP 27 2006-05-23... Complete Deskt...

900 60 001 3000 PP 12 NULL someitem

500 60 DPF78 4000 TT 2 2006-07-23... Laser printer

Which includes order\_no 900, which isn’t the biggest sale to company 3000. With the current code any sale of value 2, 7, 12 or 27 will be selected regardless.

Let's leave the extra sale in the table for now, so that we can be certain later that we get the right solution.

### 6 Moving towards the solution

Where we went wrong was:

Introducing a GROUP BY clause to the subquery hence producing multiple values. We should have:

Introduced a WHERE clause that found the biggest of some rows, not the biggest of all 8. That WHERE clause has to be based on company\_no.

So we are now at this point:

SELECT \*

FROM sale

-- safe as subquery will return one value only

WHERE order\_value =

(

SELECT MAX(order\_value)

FROM sale

-- this line added, no grouping!

WHERE company\_no = ??)

ORDER BY company\_no

If we change the '??' to say 1000 we will then get the biggest sale to company 1000 (1 row).

If we change the '??' to say 2000 we will then get the biggest sale to company 2000 (1 row).

But, we want both sales to appear together along with the biggest for every other company\_no. So, as we can't hard code the value we better say company\_no, arriving at this point:

SELECT \*

FROM sale

-- safe as subquery will return one value only

WHERE order\_value =

(

SELECT MAX(order\_value)

FROM sale

WHERE company\_no = company\_no

)

ORDER BY company\_no

However, as you can imagine a line of code saying:

WHERE company\_no = company\_no

Says, "include in the calculation every row whose company\_no column contains the same data as it's company\_no column", rather true for every row!!

The line

WHERE company\_no = company\_no

is about as useful as WHERE 7 = 7 and 65 = 65!

But we are now very close.

### 7 Mandatory aliasing needed and solution achieved

Recognise this: There are 2 different queries of 'sale' happening and we need to keep taking/using the 'company\_no' from the sale row being read by the outer query that is itself constantly changing as we browse the table.

The problem is the SQL parser will check and resolve column names (requiring that they are unambiguous) against the NEAREST 'FROM' clause, which for our dodgy line

WHERE company\_no = company\_no

means against the sale (query) of the 'FROM' clause immediately above it (the inner query).

The solution to the problem is: (this code runs and works)

SELECT \*

-- note definition of alias

FROM sale As OuterSaleBrowse

WHERE order\_value =

(

SELECT MAX(order\_value)

FROM sale

-- note use of alias

WHERE company\_no = OuterSaleBrowse.company\_no

)

ORDER BY company\_no

This produces:

ord emp their cmpy cntct ord order desc

no no ord no no code val date

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

100 60 AA1 1000 MM 7 2006-06-24... Toshiba 6700 Pro

300 60 Ord39 2000 OO 12 2006-07-14... ScanPRO 4800 S...

600 50 AC12 3000 PP 27 2006-05-23... Complete Deskt...

500 60 DPF78 4000 TT 2 2006-07-23... Laser printer

The correct answer, hurrah! Even with that 9th sale of 12 to company\_no 3000 in the data.

**Note:** The rules of algebra just went out the window, things in parentheses don't necessarily run 1st and just once! This subquery cannot run standalone. You’d get the following error message:

"The column prefix 'OuterSaleBrowse' does not match with a table name or alias name used in the query."

This is called a CORRELATED subquery, meaning "the subquery requires a value from above rather than (the usual) value from below".

The table of the inner query DID NOT NEED TO BE ALIASED. However, the table of the outer query MUST BE ALIASED.

Common practice (and what every textbook does) is to alias both tables (one for each query made against the table):

### 8 Tidying up the solution

SELECT \*

-- simple alias 'S1' (required)

FROM sale S1

WHERE order\_value =

(

SELECT MAX(order\_value)

-- simple alias, not needed

-- but makes next line better

FROM sale S2

-- note use of both alias'

WHERE S2.company\_no = S1.company\_no

)

ORDER BY company\_no

ord emp their cmpy cntct ord order desc

no no ord no no code val date

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

100 60 AA1 1000 MM 7 2006-06-24... Toshiba 6700 Pro

300 60 Ord39 2000 OO 12 2006-07-14... ScanPRO 4800 S...

600 50 AC12 3000 PP 27 2006-05-23... Complete Deskt...

500 60 DPF78 4000 TT 2 2006-07-23... Laser printer

### 9 Correlated queries are all the same

Once you get the hang of it, correlated queries are all the same. Let’s get the biggest per emp\_no, or biggest per contact

In 20 seconds flat you can copy/paste the query and change

WHERE S2.company\_no = S1.company\_no

to

WHERE S2.emp\_no = S1.emp\_no

and change ORDER BY to be based on 'emp\_no' and you have the biggest sale per employee

SELECT \*

-- simple alias 'S1' (required)

FROM sale S1

WHERE order\_value =

(

SELECT MAX(order\_value)

-- simple alias, not needed

-- but makes next line better

FROM sale S2

-- note use of both alias'

WHERE S2.emp\_no = S1.emp\_no

)

ORDER BY emp\_no

ord emp their cmpy cntct ord order desc

no no ord no no code val date

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

400 10 DGG5 1000 MM 5 2006-08-09... Modems and Cab...

600 50 AC12 3000 PP 27 2006-05-23... Complete Deskt...

300 60 Ord39 2000 OO 12 2006-07-14... ScanPRO 4800 S...

900 60 001 3000 PP 12 NULL someitem

Note, employee 60 has made 2 sales of 12 which are the Joint highest for that employee.

The following simple change finds the biggest sale per contact code.

SELECT \*

FROM sale S1

WHERE order\_value =

(

SELECT MAX(order\_value)

FROM sale S2

WHERE S2.contact\_code = S1.contact\_code

)

ORDER BY contact\_code

They are all the same!!

So can we agree: "it runs the subquery once for each row of the outer query to generate a value to then decide whether that row of the outer browse is SELECTed".

### 10 The ALTERNATIVE: a subquery in the FROM clause AKA “an inline VIEW!!”

The process above could well be be inefficient if the table contained 10000 rows, because the subquery would need to run 10000 times.

Fortunately, the developers who write DBMSystems and Execution engines realised a short cut can be made:

Assuming we want the “biggest per company” we could generate a temporary summarised table containing one row for each company, say, holding just the 'company\_no' and the biggest 'order\_value' for that company.

It would look like this:

company\_no biggest

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

1000 7

2000 12

3000 27

4000 2

They then JOIN the original 10000 row table to the summarised table on both columns ('company\_no' and 'order\_value') to produce the result set.

You could do this yourself by

1. Creating (using CREATE Table) a Summary\_Table.
2. INSERTing into the Summary\_Table the result of a SELECT that did a GROUP BY company\_no.
3. Then write your JOIN.

However, it is easier than that because all the major players allow you to put a subquery into the 'FROM' clause, doing what is known as an “inline VIEW” - a View that is non-catalogued (not in the schema).

These next two queries therefore achieve the same thing. The first one is correlated (and we’ve seen it before):

SELECT \*

FROM sale S1

WHERE order\_value =

(

-- Standard correlated sub query

SELECT MAX(order\_value)

FROM sale S2

WHERE S2.company\_no = S1.company\_no

)

ORDER BY company\_no

The second one is not CORRELATED. It creates the temporary summarised table in the FROM clause:

SELECT \*

FROM sale S JOIN

(

SELECT company\_no, MAX(order\_value) AS 'biggest'

FROM sale

GROUP BY company\_no

-- inline query must be aliased for 'on' to work

) AS SUMM

-- JOINing on both columns

ON S.company\_no = SUMM.company\_no

AND S.order\_value = SUMM.biggest

ORDER BY S.company\_no

It produces the correct answer, of course!

ord emp their cmpy cntct ord order desc

no no ord no no code val date

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

100 60 AA1 1000 MM 7 2006-06-24... Toshiba 6700 Pro

300 60 Ord39 2000 OO 12 2006-07-14... ScanPRO 4800 S...

600 50 AC12 3000 PP 27 2006-05-23... Complete Deskt...

500 60 DPF78 4000 TT 2 2006-07-23... Laser printer

This is the end of the tutorial. You can try it with some Lab work NOW.

### Exercise 10

Use either or both (preferably) techniques to display the biggest sale per dept\_no.

One potential problem is that the 'sale' table doesn’t have a 'dept\_no' column, but knows someone who does.

**Note:** There is no need to access the table called dept!

Either start from a 'blank page' or use a 'starter' template (see next page) and 'fill in the gaps'.

Starter Template for 'correlated version'

-- display dept\_no to make sense of output SELECT

SELECT dept\_no, S1.\*

FROM sale S1

WHERE order\_value =

(

SELECT MAX(order\_value)

FROM sale S2

-- must be correlated on dept\_no

WHERE ?.dept\_no = ?.dept\_no

)

ORDER BY dept\_no

If you do it right then the results should look like the following (2 rows):

dept ord emp their cmpy cntct ord order desc

no no no ord no no code val date

-----.---- ---- ------- ------ ----- ---- ------------- ----------

1 400 10 DGG5 1000 MM 5 2006-08-09... Modems a...

3 600 50 AC12 3000 PP 27 2006-05-23... Complete...

Starter Template for 'inline VIEW version'

SELECT dept\_no, S.\*

FROM sale S

??

JOIN

(

SELECT dept\_no, MAX(order\_value) AS 'biggest'

FROM sale

??

GROUP BY dept\_no

) ?? -- inline query aliased

-- JOIN on both columns

ON ?.dept\_no = ?.dept\_no

AND S.order\_value = ??.biggest

ORDER BY dept\_no

If you do it right then the results should look like the following (2 rows):

dept ord emp their cmpy cntct ord order desc

no no no ord no no code val date

-----.---- ---- ------- ------ ----- ---- ------------- ----------

1 400 10 DGG5 1000 MM 5 2006-08-09... Modems a...

3 600 50 AC12 3000 PP 27 2006-05-23... Complete...

**Hints:** The 'inline VIEW' version should effectively JOIN 2 “virtual” tables on dept\_no and order\_value. Try writing the code that creates each “table” separately.

The first few lines of the outer query (those before the first “JOIN”) if run in isolation should produce the following output (the asterisks will be explained in a minute):

dept ord emp their cmpy cntct ord order desc

no no no ord no no code val date

-----.---- ---- ------- ------ ----- ---- ------------- ----------

3 100 60 AA1 1000 MM 7 2006-06-24... Toshiba...

3 200 0 Ord34 3000 QQ 6 2006-05-01... MS Offic...

3 300 60 Ord39 2000 OO 12 2006-07-14... ScanPRO ...

1 \*\*\* 400 10 DGG5 1000 MM 5\*\*\* 2006-08-09... Modems a...

3 500 60 DPF78 4000 TT 2 2006-07-23... Laser pr...

3 \*\*\* 600 50 AC12 3000 PP 27\*\*\*2006-05-23... Complete...

1 700 10 23 2000 OO 3 2006-01-23... SQL Serv...

3 800 60 B-123E 3000 RR 3 2006-11-15... Printer ...

3 900 60 001 3000 PP 12 NULL someitem

The output of the subquery appearing within the first JOIN should be:

dept\_no biggest

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

1 \*\*\* 5 \*\*\*

3 \*\*\* 27 \*\*\*

The two queries should be JOINed on the rows WHERE the asterisks show matching values.

### Exercise 11

You know more than one technique of finding depts with no people. The same techniques can be used to find people with no sales. This should be quite easy because there is a direct link between the 2 tables involved.

But what if you wanted to find depts with no sales? The tables have no common column.

Are we trying to put the dept row in the output if there is no (sounds like SQL - NOT EXISTS?) answer set (sounds like SELECT \*?) when we go looking for sales made by any person in that dept (dept\_no)"?

Earlier you learnt that a subquery that follows an EXISTS (or NOT EXISTS) clause can SELECT anything (it doesn’t matter what) because a TRUE or FALSE is 'returned'.

But that was at a time when you believed all subqueries ran just once. We now know that this is NOT the case if the subquery is correlated.

So for example it is TRUE that when you search the sales table for an emp\_no whose dept\_no is 2 an answer set cannot be created because there is no dept\_no with a value of 2 in the table. I.E the answer set cannot EXIST

It is NOT TRUE (i.e. FALSE) that there does not exist an answer set when you search for sales by an emp\_no whose dept\_no is 1.

So, using these ideas try to find the departments that have no associated sales by converting the following into a correlated NOT EXISTS.

SELECT \* -- show me the details of

FROM dept D -- a dept (known as D)

WHERE NOT EXISTS -- if there isn’t

(

SELECT \* -- an answer set

FROM sale -- when I look for sales

?? -- whose salesperson's

WHERE dept\_no = ? -- dept\_no equals the

-- dept\_no of the dept(D)

)

If you do it right then the results should look like the following (2 rows):

dept dept manager sales

no name target

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

2 Business Systems Barbara Banana 15.00

4 Desktop Systems Diver Dan 5.00

5 Electrical Repairs Xavier Xylophone 45.00

### Exercise 12

But surely this next non-correlated alternative is much easier

1. to read
2. to write
3. to maintain

SELECT \*

FROM dept

-- whose dept\_no isn’t one of the dept\_no's

WHERE dept\_no NOT IN

(

-- of the people who have sold

SELECT dept\_no

FROM sale S JOIN salesperson SP

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

)-- not correlated

The only problem is if the employee with an emp\_no of 10 leaves the company and you run this UPDATE:

UPDATE salesperson

SET dept\_no = NULL

WHERE emp\_no = 10

It all goes wrong because the subquery:

SELECT dept\_no

FROM sale S JOIN salesperson SP

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

now produces a result set of:

dept\_no

-----------

3

3

3

NULL -- a sale made by salesperson 10

3

3

NULL -- a sale made by salesperson 10

3

3

**Question:** Is the value '1' as in 'dept\_no = 1' NOT IN that result set?

You have 3 choices:

1. TRUE (it is NOT IN)
2. FALSE (it is IN)
3. not sure/unknown/NULL (I am not sure if 1 is NOT IN)

The answer(unfortunately) is ‘c’.

So this query:

SELECT \*

FROM dept

-- whose dept\_no isn’t one of the dept\_no's

WHERE dept\_no NOT IN

(

-- of the people who have sold

SELECT dept\_no

FROM sale S JOIN salesperson SP

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

)

produces this:

(0 row(s) affected)

If any of the dept\_nos of the salespeople who have sold is NULL.

**Note:** If you were to add a suitable 'WHERE' to the subquery it will work again, but would you remember.

SELECT \*

FROM dept

-- whose dept\_no isn’t one of the dept\_no's

WHERE dept\_no NOT IN

(

-- of the people who have sold

SELECT dept\_no

FROM sale S JOIN salesperson SP

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

-- but ignoring sales made by a

-- person not in any dept

WHERE SP.dept\_no IS NOT NULL

)

Maybe the decision in the future is to set the emp\_no of the salesperson to NULL on sale rows that he was responsible for if he leaves the company and we remove his row from salesperson.

(Foreign Key constraints can actually automate this via 'ON DELETE SET NULL')

A more sensible option of course would be to set his employment\_status column to 'L' meaning "has 'L'eft" the company.

So just in case salesperson is ever going to be NULL in the future you should recognise it is safer to use the NOT EXISTS which still returns TRUE if one of the dept\_no's is NULL.

SELECT \*

FROM dept D

WHERE NOT EXISTS

(

SELECT \*

FROM sale S JOIN salesperson SP

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

WHERE SP.dept\_no = D.dept\_no

)

Produces those depts with no sales now that emp\_no 10 (originally in dept\_no 1) has left dept\_no=1 and moved to dept\_no=NULL.

dept dept manager sales

no name target

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

1 Animal Products Adam Apricot 5.00

2 Business Systems Barbara Banana 15.00

4 Desktop Systems Diver Dan 5.00

5 Electrical Repairs Xavier Xylophone 45.00

clean up:

UPDATE salesperson

SET dept\_no = 1

WHERE emp\_no = 10

DELETE FROM sale WHERE order\_no = 900