

Return to Home Page

April 2000



Creating SQL Results Sets with Aggregate and Detail Data

By Craig S. Mullins

Every SQL manipulation statement operates on a table and results in another table. All operations native to SQL, therefore, are performed at a set level. One retrieval statement can return multiple rows; one modification statement can modify multiple rows. This feature of relational databases is called *relational closure*. Relational closure is the major reason that relational databases such as SQL Server generally are easier to maintain and query than other types of databases. Additionally, SQL specifies what data to retrieve or manipulate, but does not specify how to accomplish these tasks. This keeps SQL intrinsically simplistic.

If you can remember the set-at-a-time orientation of SQL Server, and that your queries specify "what" to retrieve and not "how" to retrieve it, then you grasp the essence and nature of SQL. The capability to act on a set of data coupled with the lack of need for establishing *how* to retrieve and manipulate data defines SQL as a non-procedural language.

SQL Server also provides a series of functions that can be used to aggregate data. Certain column functions can be used to compute, from a group of rows, a single value for a designated column or expression. For example, the **sum** function can be used to add values in multiple rows, returning the sum of the values instead of each individual value.

Details and Aggregates

Now, taking the previous discussion into consideration, what if you need to return detail data and aggregated data in the same results set using SQL. For example, consider the sample *pubs* database that comes with Microsoft SQL Server. The sales table contains information about each title sold (refer to Figure 1 for a sample of the data contained in the sales table).

🛛 🔻 Action 🔻 View 💌 Tools					⊨⇒ 🖻 🕽	x 🔊 🛱	bases\pubs\Tabl	. 🔂 🔁 🕏 🗣	
Console Root	_	(r	Name	(c	Iwner	Туре	Create Date		·
🗄 📲 Microsoft SQL Servers			authors	d	bo	User	1998-06-19		
🗄 🍯 SQL Server Group			i discounts		dbo	User 1998-06-1	1998-06-19		
📄 🔂 GARGANTUA (Win		: 95/98)	employee jobs	đ	ьо	User	1998-06-19		
👘 Distributed Tran		on Coordin 📑	iobs	d	ьо	User	1998-06-19		
🗄 🔂 SQL Server Age	ent		pub_info	d	ьо	User	1998-06-19		
- 🔁 SQL Mail			publishers		bo	User	1998-06-19		
- Cogins			roysched		bo	User	1998-06-19		
Server Roles			sales		bo	User	1998-06-19		
Remote Servers						0001	.0000010		
⊡ 🕮 Linked Servers ⊡ 🧰 Databases	10 2	:Data in Ta	ble 'sales'						
± Batabases	ß	🔤 💷 SC	I 🔲 🗗 !	🗖 💱 🛓 🏅	↓ 🛠 🔚	+			
	_					-			
Datab.		stor_id	ord_num	ord_date	qty		payterms	title_id	
Datab.		6380	6871	9/14/94	5		Net 60	BU1032	
	6	6380	722a	9/13/94	3		Net 60	PS2091	
	- 1							0,00000	
		7066	A2976	5/24/93	50		Net 30	PC8888	
- 🧰 SQL S	7	7066	QA7442.3	9/13/94	75		ON invoice	P52091	
SQL S Stored	777	7066 7067	QA7442.3 D4482	9/13/94 9/14/94	75 10		ON invoice Net 60	P52091 P52091	
	7777	7066	QA7442.3	9/13/94	75		ON invoice	P52091	
	777777777777777777777777777777777777777	7066 7067 7067	QA7442.3 D4482 P2121	9/13/94 9/14/94 6/15/92	75 10 40		ON invoice Net 60 Net 30	PS2091 PS2091 TC3218	
	777777777777777777777777777777777777777	7066 7067 7067 7067	QA7442.3 D4482 P2121 P2121	9/13/94 9/14/94 6/15/92 6/15/92	75 10 40 20		ON invoice Net 60 Net 30 Net 30	P52091 P52091 TC3218 TC4203	
SQLS Stored Defaul User Datab.	777777	7066 7067 7067 7067 7067	QA7442.3 D4482 P2121 P2121 P2121 P2121	9/13/94 9/14/94 6/15/92 6/15/92 6/15/92	75 10 40 20 20		ON invoice Net 60 Net 30 Net 30 Net 30	P52091 P52091 TC3218 TC4203 TC7777	
	777777777777777777777777777777777777777	7066 7067 7067 7067 7067 7067 7131 7131 7131	QA7442.3 D4482 P2121 P2121 P2121 N914008 N914014 P3087a	9/13/94 9/14/94 6/15/92 6/15/92 6/15/92 9/14/94 9/14/94 5/29/93	75 10 40 20 20 20 25 25 20		ON invoice Net 60 Net 30 Net 30 Net 30 Net 30 Net 30 Net 60	P52091 P52091 TC3218 TC4203 TC7777 P52091 MC3021 P51372	
SQL S Stored Defaul User C Jatab.	777777777777777777777777777777777777777	7066 7067 7067 7067 7067 7131 7131 7131 7131	QA7442.3 D4482 P2121 P2121 P2121 N914008 N914014 P3087a P3087a	9/13/94 9/14/94 6/15/92 6/15/92 9/14/94 9/14/94 5/29/93 5/29/93	75 10 20 20 20 25 20 25 20 25		ON invoice Net 60 Net 30 Net 30 Net 30 Net 30 Net 30 Net 30 Net 60	P52091 P52091 TC3218 TC4203 TC7777 P52091 MC3021 P51372 P52106	
Stored Stored Defaul User C Datab B M SQL Server Lc Datab C Datab Datab Datab Datab Datab	777777777777777777777777777777777777777	7066 7067 7067 7067 7067 7131 7131 7131 7131 7131 7131	QA7442.3 D4482 P2121 P2121 P2121 N914008 N914014 P3087a P3087a P3087a	9/13/94 9/14/94 6/15/92 6/15/92 9/14/94 9/14/94 5/29/93 5/29/93 5/29/93	75 10 40 20 20 20 25 20 25 20 25 15		ON invoice Net 60 Net 30 Net 30 Net 30 Net 30 Net 30 Net 30 Net 60 Net 60 Net 60	PS2091 PS2091 TC3218 TC4203 TC7777 PS2091 MC3021 PS1372 PS2106 PS3333	
SQL S Stored Particles Defaul User C SQL Server LC Deta Transforr Backwp Devic	777777777777777777777777777777777777777	7066 7067 7067 7067 7067 7131 7131 7131 7131 7131 7131 7131 71	QA7442.3 D4482 P2121 P2121 P2121 N914008 N914014 P3087a P3087a P3087a P3087a	9/13/94 9/14/94 6/15/92 6/15/92 9/14/94 9/14/94 5/29/93 5/29/93 5/29/93 5/29/93	75 10 40 20 20 25 25 20 25 15 25		ON invoice Net 60 Net 30 Net 30 Net 30 Net 30 Net 30 Net 60 Net 60 Net 60 Net 60	PS2091 PS2091 TC3218 TC4203 TC7777 PS2091 MC3021 PS1372 PS1372 PS2106 PS3333 PS7777	
SQL S Stored Defaul User D Datab. Backup Devic Web Assistant Backup Devic	777777777777777777777777777777777777777	7066 7067 7067 7067 7131 7131 7131 7131 7131 7131 7131 71	QA7442.3 D4482 P2121 P2121 P2121 N914008 N914014 P3087a P3087a P3087a P3087a Q02299	9/13/94 9/14/94 6/15/92 6/15/92 9/14/94 9/14/94 5/29/93 5/29/93 5/29/93 5/29/93 10/28/93	75 10 40 20 20 25 25 25 15 25 15 25 15		ON invoice Net 60 Net 30 Net 30 Net 30 Net 30 Net 30 Net 60 Net 60 Net 60 Net 60	PS2091 PS2091 TC3218 TC4203 TC7777 PS2091 MC3021 PS105 PS2106 PS3106 PS3333 PS7777 BU7852	
Stored Stored Parkes Defaul User C Backup Devic Backup Devic Web Assistant Database Mair Assistant	777777777777777777777777777777777777777	7066 7067 7067 7067 7067 7131 7131 7131 7131 7131 7131 7131 71	QA7442.3 D4482 P2121 P2121 P2121 N914008 N914014 P3087a P3087a P3087a P3087a P3087a Q02299 TQ455	9/13/94 9/14/94 6/15/92 6/15/92 9/14/94 9/14/94 5/29/93 5/29/93 5/29/93 10/28/93 12/12/93	75 10 40 20 20 25 20 25 20 25 15 15 25 15 15 15		ON invoice Net 60 Net 30 Net 30 Net 30 Net 30 Net 30 Net 60	PS2091 PS2091 TC4218 TC4203 TC4777 PS2091 MC3021 PS1372 PS1372 PS1373 PS7777 BU7832 MC2222	
SQL S Stored Defaul User D Datab. Backup Devic Web Assistant	777777777777777777777777777777777777777	7066 7067 7067 7067 7067 7131 7131 7131 7131 7131 7131 7131 71	QA7442.3 D4482 P2121 P2121 P2121 N914008 N914014 P3087a P3087a P3087a P3087a Q02299 TQ456 X999	9/13/94 9/14/94 6/15/92 6/15/92 6/15/92 9/14/94 5/29/93 5/29/93 5/29/93 5/29/93 10/28/93 12/12/93 12/12/93	75 10 40 20 20 25 20 25 15 25 15 15 10 35		ON invoice Net 30 Net 30 Net 30 Net 30 Net 30 Net 30 Net 60 ON invoice	PS2091 PS2091 TC3218 TC4203 TC7777 PS2091 MC3021 PS1372 PS1372 PS2106 PS1373 PS2106 PS3333 PS2106 PS3333 PS7777 BU7832 MC2222 BU2075	
Stored Stored Paules Defaul Stored Defaul Stored User C Stored Database Stored Stor	77777777777777777777777777777777777777	7066 7067 7067 7067 7067 7067 7131 7131 7131 7131 7131 7131 7131 71	QA7442.3 D4462 P2121 P2121 P2121 P2121 P2121 P2121 P2121 P2121 P2087a P3087a P3087a P3087a P3087a Q2229 TQ456 X3999 423U922	9/13/94 9/14/94 6/15/92 6/15/92 9/14/94 9/14/94 9/14/94 5/29/93 5/29/93 5/29/93 10/28/93 10/28/93 12/12/93 2/21/93 9/14/94	75 10 40 20 25 20 25 25 15 25 15 25 15 15 10 35 15		ON invoice Net 60 Net 30 Net 30 Net 30 Net 30 Net 30 Net 30 Net 60 ON invoice ON invoice	PS2091 PS2091 TC3218 TC4203 TC7777 PS2091 MC3021 PS105 PS2106 PS3333 PS7777 BU7832 MC2222 BU2075 BU2075 BU2075	
Stored Stored Paules Defaul Stored Defaul Stored User C Stored Database Stored Stor	77777777777777777777777777777777777777	7066 7067 7067 7067 7067 7131 7131 7131 7131 7131 7131 7131 71	QA7442.3 D4482 P2121 P2121 P2121 N914008 N914014 P3087a P3087a P3087a P3087a Q02299 TQ456 X999	9/13/94 9/14/94 6/15/92 6/15/92 6/15/92 6/15/92 9/14/94 9/14/94 5/29/93 5/29/93 5/29/93 5/29/93 10/28/93 12/12/93 2/21/93 9/14/94	75 10 40 20 20 25 20 25 15 25 15 15 10 35		ON invoice Net 30 Net 30 Net 30 Net 30 Net 30 Net 30 Net 60 ON invoice	PS2091 PS2091 TC3218 TC4203 TC7777 PS2091 MC3021 PS1372 PS1372 PS2106 PS1373 PS2106 PS3333 PS2106 PS3333 PS7777 BU7832 MC2222 BU2075	
SQL S Stored Defaul User C Datab. Backup Devic Backup Devic Web Assistant Database Mair		7066 7067 7067 7067 7131 7131 7131 7131 7131 7131 7131 71	QA7442.3 D4462 P2121 P2121 P2121 N914008 P3087a P3097a P3087a P30970 P30970 P30970 P30	9/13/94 9/14/94 6/15/92 6/15/92 9/14/94 9/14/94 9/14/94 5/29/93 5/29/93 5/29/93 10/28/93 10/28/93 12/12/93 2/21/93 9/14/94	75 10 20 20 25 25 25 15 15 15 10 35 15 15		ON invoice Net 60 Net 30 Net 30 Net 30 Net 30 Net 60 Net 60 Net 60 Net 60 ON invoice ON invoice ON invoice	PS2091 PS2091 TC4218 TC4203 TC4777 PS2091 MC3021 PS1372 PS106 PS333 PS3333 PS7777 BU7832 MC2222 BU2075 MC3021 BU1032	

Figure 1. Sample sales table data.

To return a simple list of the detail information contained in this table you could issue the following query:

But what if, instead of just returning qty on each row, you also wished to return the average quantity, maximum quantity, and minimum quantity. This is not quite so easy. You can use the **avg**, **max**, and **min** functions to return the average, maximum, and minimum qty of a sale by store as follows:

```
select stor_id, avg(qty), max(qty),
min(qty)
from sales
group by stor_id
```

This query returns a single row for each store containing the average, maximum, and minimum sale. It does not contain any detail information, only aggregated information.

However, there are times when it is desirable for a single query to return both detail and aggregate information. For example, what if you wished to compare each sale qty to the average sale qty? There are several ways to do this, but it is desirable to be able return both values in a single query to simplify your application.

One solution is to create a view. Consider the following view named sales_qty created on the sales table, as shown here:

```
use pubs
create view sales_qty
   (stor_id, max_qty, min_qty, avg_qty)
as select stor_id, avg(qty), max(qty),
min(qty)
from sales
group by stor id
```

After the view is created, you can issue the following **select** statement joining the view to the base table, thereby providing both detail and aggregate information on each report row:

```
select s.stor_id, s.title_id, s.qty,
        q.max_qty, q.min_qty, q.avg_qty
from sales s,
        sales_qty q
where s.stor_id = q.stor_id
order by 1
```

This query returns one row for each sale, listing both the store identifier and title identifier, along with the actual quantity, and the maximum, minimum, and average quantity for that store. The data will be returned in stor_id order. This can be changed simply by changing the **order by** clause to specify which column the results should be sorted by.

Additionally, it is important to remember the meaning of the aggregates returned by the view. The aggregated data is by store, and not by title. Neither does the aggregated data represent the average, minimum, and maximum of all sales (independent of store). This is because that is the way the view was created. Do not forget the definition of the view. View definitions can be retrieved from the SQL Server system catalog tables using the **sp_helptext** stored procedure as follows:

use pubs

```
exec sp_helptext 'sales_qty'
```

This will return the actual SQL DDL that was used to create the view.

If additional information, such as the publication title is required, you can add the title table to the join as follows:

```
select s.stor_id, s.title_id, t.title,
s.qty,
        q.max_qty, q.min_qty, q.avg_qty
from sales s,
        sales_qty q,
        titles t
where s.stor_id = q.stor_id
and s.title_id = t.title_id
order by 1
```

Indeed, once the view is created, you can use it in any way allowable by SQL Server. Using views in this manner you can create any number of detail and aggregate combination reports that your organization may desire.

Other Approaches

Of course, this is only one example of how to return detail and aggregate data in a single row. You could also use a temporary table. A temporary table is created by prefacing the table name (in the create statement) with a number sign (#). Temporary tables are stored in the temporary database, **tempdb**. The **tempdb** database is used to hold all temporary tables and any other temporary storage needs. It is a global resource; the temporary tables and stored procedures for all users connected to the system are stored there. SQL Server re-creates **tempdb** every time it is started. This clears out data so the system starts with a clean copy of **tempdb**. Since **tempdb** is be definition, temporary, this is not a problem.

So, you could create a table in **tempdb** as follows:

```
use pubs
create table #sales_qty
(
    stor_id char(4),
    max_qty smallint,
    min_qty smallint,
    avg_qty smallint
)
```

Then you would have to populate the table with data using a bulk **insert**, for example:

```
insert into #sales_qty
  (select stor_id, avg(qty), max(qty),
min(qty)
  from sales
  group by stor id)
```

Then you can run the query to retrieve the detail information from sales and the aggregate information from the temporary table as follows:

```
select s.stor_id, s.title_id, s.qty,
        q.max_qty, q.min_qty, q.avg_qty
from sales s,
        #sales_qty q
where s.stor_id = q.stor_id
order by 1
```

Views, however, are easier to implement than this method. The advantage of a view over a temporary table is that the view does not require storage. Once created, it is always in sync with the base table(s) that it is defined on. The temporary table will consume storage, but only temporarily. Temporary tables are automatically dropped on disconnect. So even though you will need to create the temporary table only once, you will need to populate it with the aggregated data every time you run a query that needs the information. Furthermore, when tempdb is used you will need to concern yourself with the possibility that the data you need to store in temporary tables exceeds the amount currently allocated to tempdb. For these reasons, views are preferred over temporary tables.

Another method, in this case preferable to the view implementation, is to use in-line views. SQL Server supports the capability to code a **select** statement in the from clause of another **select** statement. This is often referred to as an in-line view because the SQL statement is placed in-line in the **from** clause of another SQL **select** statement. So, you can use an inline view to embed the aggregate query into the detail query as follows:

```
select s.stor_id, s.title_id, s.qty,
        q.max_qty, q.min_qty, q.avg_qty
from sales s,
            (select stor_id, avg(qty), max(qty),
            min(qty)
            from sales
            group by stor_id) as q
            (stor_id, avg_qty, max_qty, min_qty)
where s.stor_id = q.stor_id
order by 1
```

This returns the same results as the other methods. Refer to Figure 2 for an example of the results of this query.

SQL Server Profiler - [Untitled - SQI	. Script]			
<mark></mark> <u>F</u> ile <u>E</u> dit <u>R</u> eplay ⊻iew <u>T</u> ools <u>W</u> in	dow <u>H</u> elp			_ 8
v 🏠 💥 🕨 💶 🔳 🖽 🗟	🗐 🚡 🔨			
7 🗈 🔶 ី 🗊 🔂 *0 🕙	¢			
Event Class Text				
use pubs; select s.stor_id, s.tit from sales s, (select stor_id, from sales group by stor_id	_qty, q.avg_qty avg(qty), max(qty), mir) as q , max qty, min qty)	ı(qty)		
Event Class Event Sub Clas SQL-BatchStarting Replay Result Set	use pubs; select s.stor_id, s.title_id, stor_id, title_id, gty, max_gty, min_c	s.qtv, q.max. qtv, q.min. qtv, q.avg. qtv from sales (v.avg. qtv		max(qty), min(qty) from sales group by stor_it
SQL:BatchStarting Replay Result Set Replay Result Row	use pubs; select s.stor_id, s.title_id stor_id, title_id, qty, max_qty, min_c stor_id = 6380, title_id = BU1032,	<mark>ty, avg_qty</mark> qty = 5, max_qty = 5, min_qty = 3, avg_qty = 4		max(qty), min(qty) from sales group by stor, is
SQL:BatchStarting Replay Result Set Replay Result Row Replay Result Row Replay Result Row	use pubs; select s.stor_id, s.title_id stor_id, title_id, qty, max_qty, min_c stor_id = 6380, title_id = BU1032, stor_id = 6380, title_id = PS2091, stor_id = 7066, title_id = PC8888,	ty, avo_qty qty = 5, max_qty = 5, min_qty = 3, avg_qty = 4 qty = 3, max_qty = 5, min_qty = 3, avg_qty = 4 qty = 50, max_qty = 75, min_qty = 50, avg_qty	= 62	max(qty), min(qty) from sales group by stor_ii
SQL:BatchStarting Replay Result Set Replay Result Row Replay Result Row Replay Result Row Replay Result Row	use pubs; select s.stor_id, s.title_id stor_id, title_id, qty, max_qty, min_c stor_id = 6380, title_id = BU1032, stor_id = 6380, title_id = PS2091, stor_id = 7066, title_id = PC8088, stor_id = 7066, title_id = PS2091,	ty, avg_qty ty = 5, max_qty = 5, min_qty = 3, avg_qty = 4 ty = 3, max_qty = 5, min_qty = 3, avg_qty = 4 ty = 50, avg_qty ty = 75, max_qty = 75, min_qty = 50, avg_qty ty = 75, max_qty = 75, min_qty = 50, avg_qty	= 62 = 62	max(qty), min(qty) from sales group by stor_i
SQL:BatchStarting Replay Result Set Replay Result Row Replay Result Row Replay Result Row Replay Result Row Replay Result Row Replay Result Row	use pubs; select s.stor id, s.title_id stor_id, title_id_cty, max_dty_min stor_id = 6380, title_id= BU1032, stor_id = 6380, title_id= B21031, stor_id = 7066, title_id= P52091, stor_id = 7066, title_id= P52091, stor_id = 7067, title_id= P52091, stor_id = 7067, title_id= C3218, i	by avoid y by = 5, max_qby = 5, min_qby = 3, avo_qby = 4 ty = 3, max_qby = 5, min_qby = 3, avo_qby = 4 ty = 50, max_qby = 75, min_qby = 50, avo_qby = 75, max_qby = 75, min_qby = 50, avo_qdy ty = 10, max_qby = 40, min_qby = 10, avo_qby = 40, max_qby = 40, min_qby = 11, avo_qdy = 40, max_qby = 40, min_qby = 10, avo_qdy = 40, max_qby = 10, avo_qdy = 10, avo_qdy	= 62 = 62 = 22 = 22	max(qty), min(qty) from sales group by stor_i
SQL:BatchStarting Replay Result Set Replay Result Row Replay Result Row Replay Result Row Replay Result Row Replay Result Row Replay Result Row	use pubs; select s.stor_id, s.title_id stor_id, title_id, etw.max, etw.min.c stor_id = 6380, title_id = BU1032, stor_id = 6380, title_id = PS2091, stor_id = 7066, title_id = PS2091, stor_id = 7066, title_id = PS2091, stor_id = 7067, title_id = PS2091,	by avoid y by = 5, max_qby = 5, min_qby = 3, avo_qby = 4 ty = 3, max_qby = 5, min_qby = 3, avo_qby = 4 ty = 50, max_qby = 75, min_qby = 50, avo_qby = 75, max_qby = 75, min_qby = 50, avo_qdy ty = 10, max_qby = 40, min_qby = 10, avo_qby = 40, max_qby = 40, min_qby = 11, avo_qdy = 40, max_qby = 40, min_qby = 10, avo_qdy = 40, max_qby = 10, avo_qdy = 10, avo_qdy	= 62 = 62 = 22 = 22	max(qty), min(qty) from sales group by stor_i
SQL:BatchStarting Replay Result Set Replay Result Row Replay Result Row Replay Result Row Replay Result Row Replay Result Row Replay Result Row Replay Result Row	use pubs; select s.stor id, s.title_id stor_id, title_id_cty, max_dty_min stor_id = 6380, title_id= BU1032, stor_id = 6380, title_id= B21031, stor_id = 7066, title_id= P52091, stor_id = 7066, title_id= P52091, stor_id = 7067, title_id= P52091, stor_id = 7067, title_id= C3218, i	by avoid y by = 5, max_qby = 5, min_qby = 3, avo_qby = 4 ty = 3, max_qby = 5, min_qby = 3, avo_qby = 4 ty = 50, max_qby = 75, min_qby = 50, avo_qby = 75, max_qby = 75, min_qby = 50, avo_qdy ty = 10, max_qby = 40, min_qby = 10, avo_qby = 40, max_qby = 40, min_qby = 11, avo_qdy = 40, max_qby = 40, min_qby = 10, avo_qdy = 40, max_qby = 10, avo_qdy = 10, avo_qdy	= 62 = 62 = 22 = 22	max(qty), min(qty) from sales group by stor i
SQL:BatchStarting Replay Result Set Replay Result Row Replay Result Row Seplect s.stor_id. s.title_i q.max_qty.q.min_qty rom sales s.	use pubs: select stor id. stille id stor id. Nile id. giv. max. giv. min. stor_id = 6380, bite_id = BU1032, stor_id = 6380, bite_id = BU1032, stor_id = 7066, bite_id = P52031, stor_id = 7066, bite_id = P52031, stor_id = 7067, thei_id = F52031, stor_id = 7067, thei_id = TC4208, stor_id = 7067, thei_id = TC4208, stor_id = 7067, thei_id = TC4208, stor_id = 7067, thei_id = TC4208, d. s.gty, . g.avg_gty gty). max(gty), min(gty) g g gty, min_gty)	ty avg aty aty = 5, max_qty = 5, min_qty = 3, avg_qty = 4 ty = 3, max_qty = 5, min_qty = 3, avg_qty = 4 ty = 50, max_qty = 75, min_qty = 50, avg_qty = 75, max_qty = 75, min_qty = 50, avg_qty ty = 70, max_qty = 74, min_qty = 50, avg_qty ty = 10, max_qty = 40, min_qty = 10, avg_qty ty = 40, max_qty = 40, min_qty = 110, avg_qty ty = 20, max_qty = 40, min_qty = 110, avg_qty	= 62 = 62 = 22 = 22	max(qty), min(qty) from sales group by stor

Figure 2. Using an in-line view.

This approach is the most complicated because it combines everything into a single query. However, it is also the easiest to implement once you understand the process.

Synopsis

This article has discussed three different ways to combine detail and aggregate information on a single line in a result set from a SQL Server query. Of the three, the view method is probably the easiest to understand. However, the last one, the in-line view method, is the one with the least administrative overhead because not additional SQL Server objects need to be created. Only the middle method, using temporary tables, is not recommended in practice because of the programming and administrative overhead required to implement.

From SQL Server Update (Xephon) April 2000.

© 2000 Craig S. Mullins, All rights reserved. <u>Home</u>.