SQL Programming

SQL Processing Model – Part 2

Page A-1: Intro

In a previous module we examined a model that helped explain how SQL processes the programs we write.

In this module, we extend that Processing Model to accommodate some of the new features that we've since learned.

- Set functions
- Group By
- Having

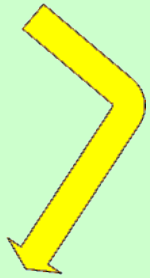
As you study this processing model keep in mind that it is only a model. A model to help you better understand how SQL works, and hence a model to help you when writing your SQL programs.

And although no single SQL implementation works in exactly this fashion, as a programmer, you can rely on them all to *behave* in this fashion.

```
SELECT title, yr
FROM movies
```

MOVIE (base table)

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	DVD	$19.95
4235	ALIEN	1986	SF	VHS	$14.35
6119	RAISING ARIZONA	1992	COM	DVD	$17.50



TITLE	YR
STAR WARS	1971
ALIEN	1986
RAISING ARIZONA	1992

Our inquiry started with the question: Can SQL process and execute program statements in a top-to-bottom fashion.

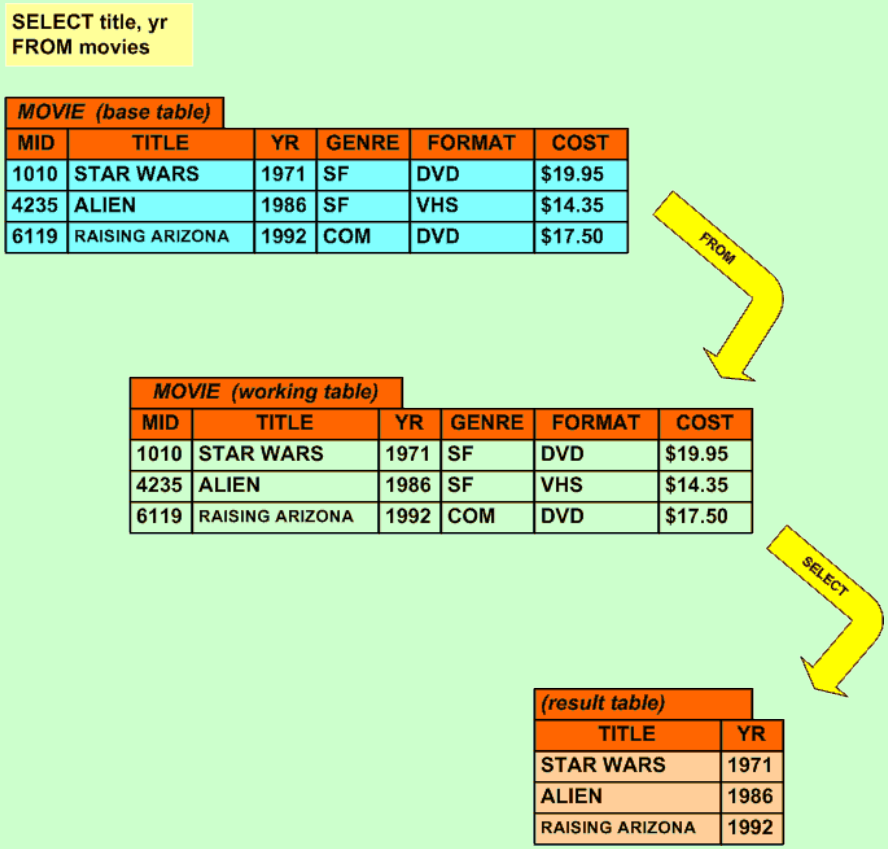
That is, can it execute the SELECT clause, and then read and execute the FROM clause, and then read and execute the WHERE clause, ….

Upon some reflection we came to realize that NO, SQL cannot sequentially process and execute statements.

SQL has no context for the columns that are listed in the SELECT clause without knowing which table(s) to use for those columns.

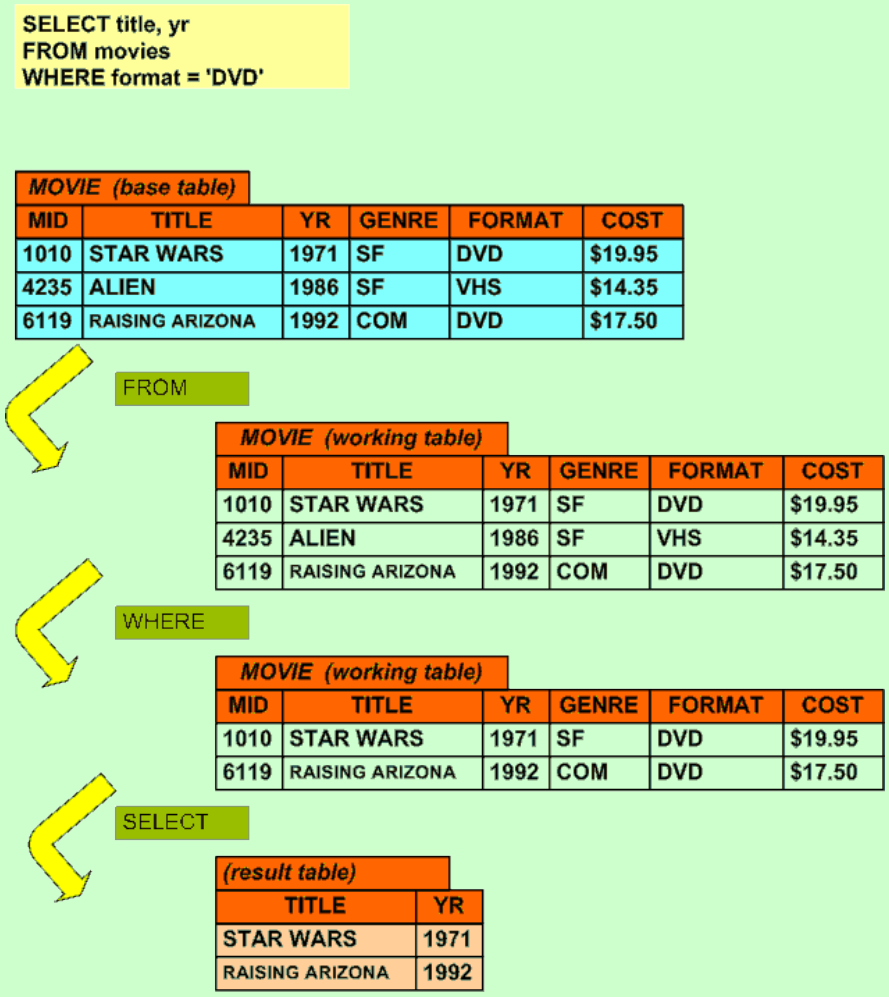
Any reference to a column must be preceded by knowledge about the table that that column resides in.

This understanding helped us develop our 1st intuition: The FROM clause is processed first.



And in this regard you can think of the FROM clause as building a working copy of the base table.

This working copy will be the foundation for all subsequent working tables, up through the creation of the final result table.



Then we considered a query such as the following one:

```
SELECT  title, yr
FROM    movie
WHERE   format = 'DVD'
```

And from here we developed our 2nd intuition that suggests that: WHERE clause processing follows the FROM clause.

The FROM clause builds a working copy of the base table(s) that are referenced in the FROM clause.

The WHERE clause then creates another intermediate working table, and this table includes only the rows that satisfy the WHERE condition.

Then the SELECT clause builds the result table by including only the columns that are named in the SELECT clause

Module 14: SQL Processing Model


```
SELECT  yr
FROM movies
WHERE format = 'DVD'
ORDER BY title
```

MOVIE (base table)

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	DVD	$19.95
4235	ALIEN	1986	SF	VHS	$14.35
6119	RAISING ARIZONA	1992	COM	DVD	$17.50

FROM

MOVIE (working table)

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	DVD	$19.95
4235	ALIEN	1986	SF	VHS	$14.35
6119	RAISING ARIZONA	1992	COM	DVD	$17.50

WHERE

MOVIE (working table)

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	DVD	$19.95
6119	RAISING ARIZONA	1992	COM	DVD	$17.50

ORDER BY

MOVIE (working table)

MID	TITLE	YR	GENRE	FORMAT	COST
6119	RAISING ARIZONA	1992	COM	DVD	$17.50
1010	STAR WARS	1971	SF	DVD	$19.95

SELECT

(result table)

YR
1992
1971

And finally, after considering the ORDER BY clause, we came to generalize that the SELECT clause is the last clause to be processed.

This SQL processing model suggests that SQL generates a number of intermediate working tables (or intermediary tables) on it's way to preparing the final result table.

In this sequencing of 'steps', the FROM clause is processed first, hence we say that the FROM clause lays the foundation for all subsequent processing

---

The WHERE clause is the next statement that is processed.

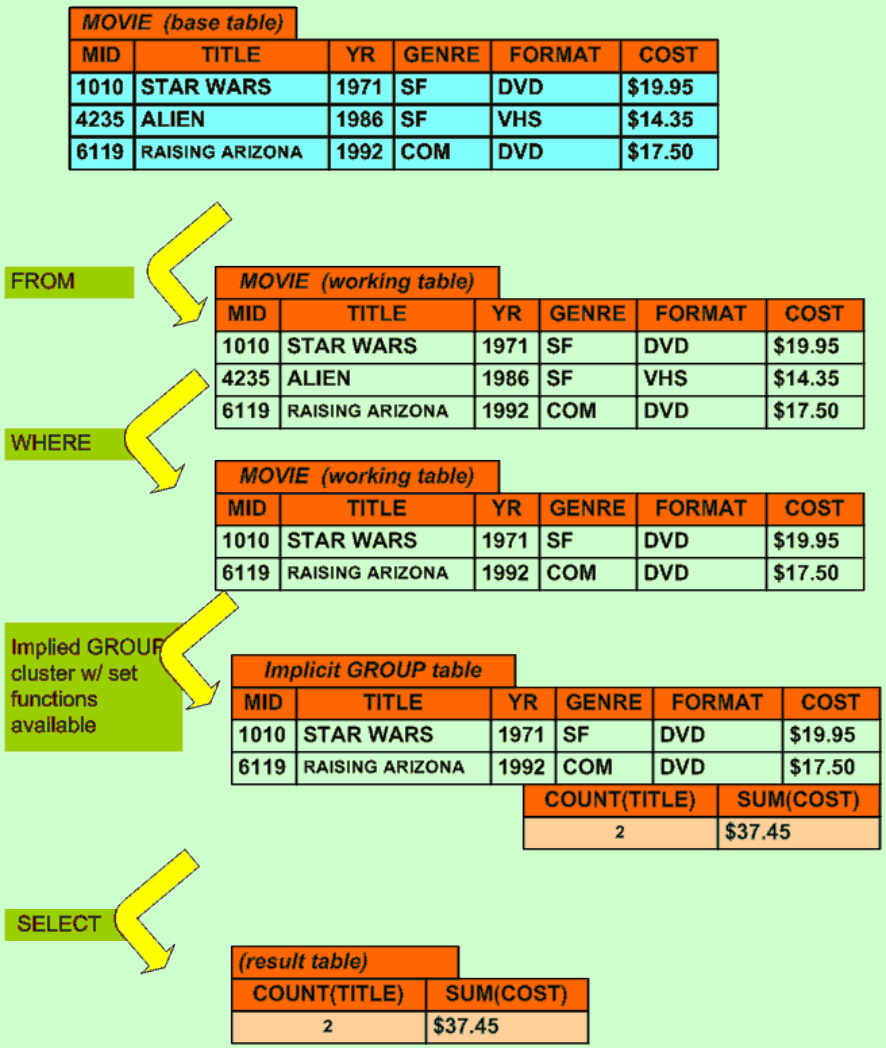
The WHERE clause tells SQL which rows in the table it should keep, and pass along to the next step.

Page B-6: SQL Processing Model

Now then, let’s pick up where we left off, and consider how SQL processes the features we just learned.

Module 14: SQL Processing Model


```
SELECT count(title), sum(cost)
FROM movies
WHERE format = 'DVD'
```



My intuitions about set functions suggests that even without a GROUP BY clause, there must be some implicit grouping or clustering of the data, so that the set functions have a 'body of data' to work with.

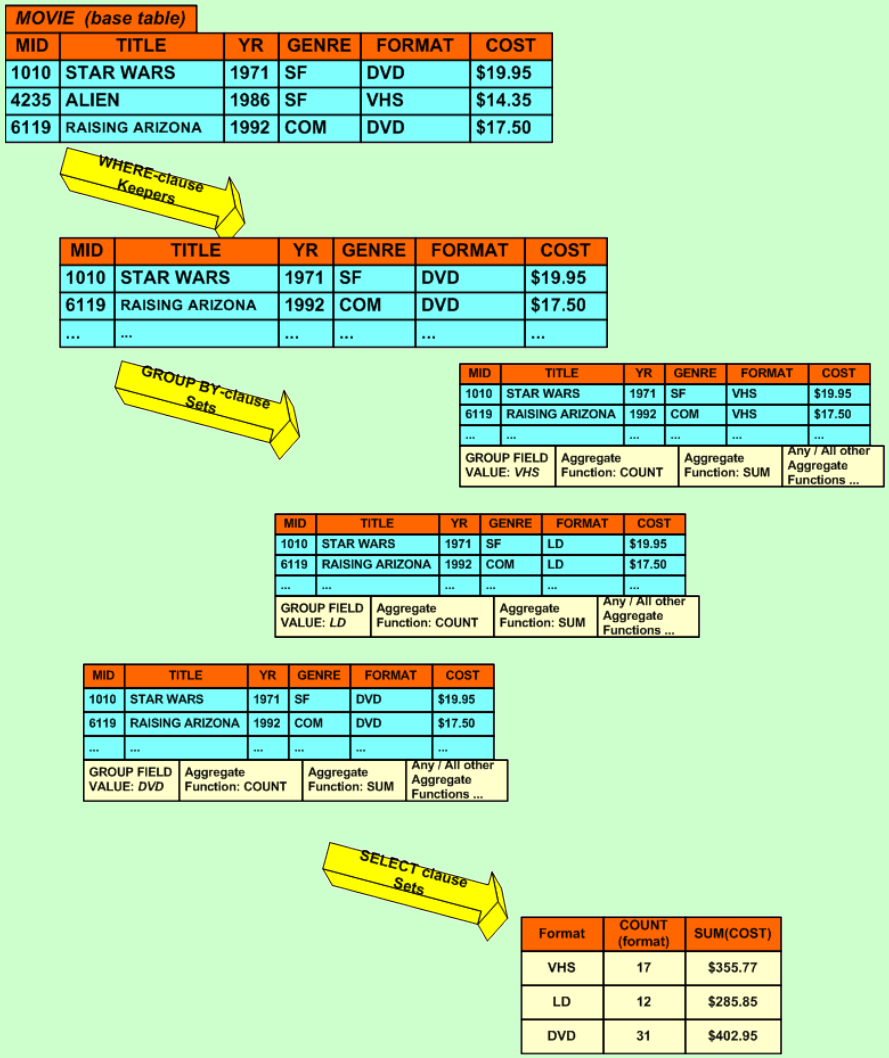
And along with this implicitly grouped body of data, all of the column functions become available.

In the diagram I try to portray the availability of the set functions with the pseudo-result table that is part of the implicit group table.

After the set functions are calculated (ie. after they become available) the remaining clauses of the program kick in (ORDER BY and then SELECT)

Module 14: SQL Processing Model

Page B-8: SQL Processing Model



This slide demonstrates how the GROUP BY clause generates working tables for each data value combination of the grouped columns.

In my mind, these tables are *explicitly* generated by the GROUP BY clause. (Remember though that this is a conceptual model of SQL processing – this ain't necessarily how it works, but it is how it behaves [sort of ☺])

In this particular example, three group cluster tables are generated, one for each of the data values that occurs in the ***format*** column of the movies table (DVD, LD, VHS).

I think of these group-tables as little 'baggies' that hold everything we can know about this group set. Once the grouping action takes place, I think of the rows in these tables as 'collapsing' into a single row - - that bottom row in each group table (the yellowish-colored one).

During Grouping

MOVIE (base table)

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	DVD	$19.95
4235	ALIEN	1986	SF	VHS	$14.35
6119	RAISING ARIZONA	1992	COM	DVD	$17.50

WHERE-clause Keepers

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	DVD	$19.95
6119	RAISING ARIZONA	1992	COM	DVD	$17.50
...	...	...	...	...	...

GROUP BY-clause Sets

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	VHS	$19.95
6119	RAISING ARIZONA	1992	COM	VHS	$17.50
...	...	...	...	...	...

GROUP FIELD VALUE: *VHS*	Aggregate Function: COUNT	Aggregate Function: SUM	Any / All other Aggregate Functions ...

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	LD	$19.95
6119	RAISING ARIZONA	1992	COM	LD	$17.50
...	...	...	...	...	...

GROUP FIELD VALUE: *LD*	Aggregate Function: COUNT	Aggregate Function: SUM	Any / All other Aggregate Functions ...

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	DVD	$19.95
6119	RAISING ARIZONA	1992	COM	DVD	$17.50
...	...	...	...	...	...

GROUP FIELD VALUE: *DVD*	Aggregate Function: COUNT	Aggregate Function: SUM	Any / All other Aggregate Functions ...

SELECT clause Sets

Format	COUNT (format)	SUM(COST)
VHS	17	$355.77
LD	12	$285.85
DVD	31	$402.95

```
SELECT format, count(format), sum(cost)
FROM movies
GROUP BY format
```

After Grouping

MOVIE (base table)

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	DVD	$19.95
4235	ALIEN	1986	SF	VHS	$14.35
6119	RAISING ARIZONA	1992	COM	DVD	$17.50

WHERE-clause Keepers

MID	TITLE	YR	GENRE	FORMAT	COST
1010	STAR WARS	1971	SF	DVD	$19.95
6119	RAISING ARIZONA	1992	COM	DVD	$17.50
...	...	...	...	...	...

GROUP BY-clause Sets

GROUP FIELD VALUE: *VHS*	Aggregate Function: COUNT	Aggregate Function: SUM	Any / All other Aggregate Functions ...

GROUP FIELD VALUE: *LD*	Aggregate Function: COUNT	Aggregate Function: SUM	Any / All other Aggregate Functions ...

GROUP FIELD VALUE: *DVD*	Aggregate Function: COUNT	Aggregate Function: SUM	Any / All other Aggregate Functions ...

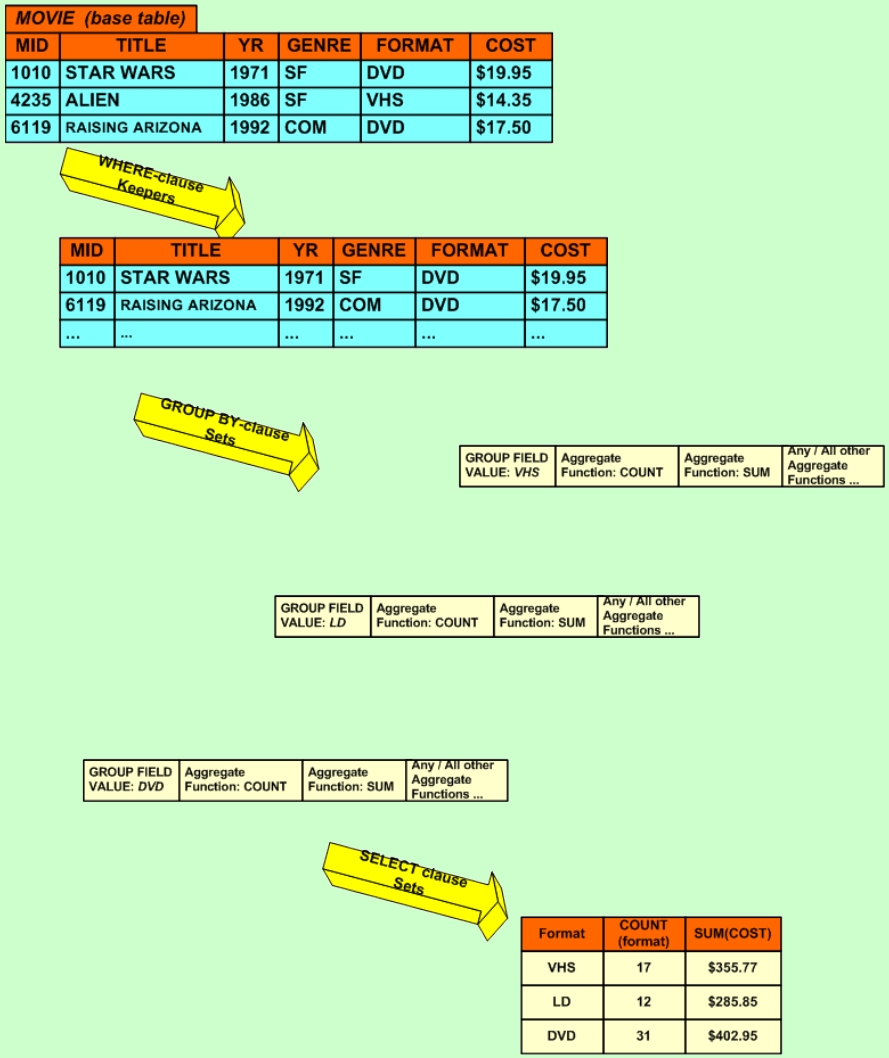
SELECT clause Sets

Format	COUNT (format)	SUM(COST)
VHS	17	$355.77
LD	12	$285.85
DVD	31	$402.95

```
SELECT format, count(format), sum(cost)
FROM movies
GROUP BY format
```

Module 14: SQL Processing Model

Page B-10: SQL Processing Model



So here's what we've got:

1) the columns that were named in the GROUP BY clause are still available, and

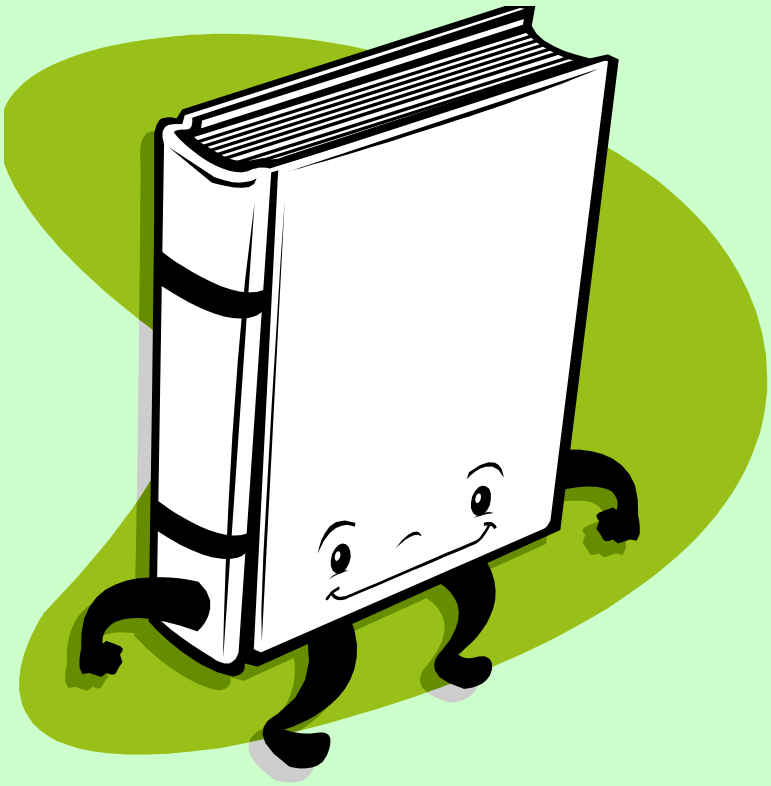
2) so are all of the aggregate functions (MIN, MAX, COUNT, ...)

These are the only columns that are available to the remaining clauses (HAVING, ORDER BY and SELECT). If your SELECT clause includes any other columns, you can expect that SQL will throw an error – something along the lines of 'Hey, you can't mix detail-level data with grouped-level data'.

It's at this stage, as the final result table is being put together, that the HAVING clause applies. Only those result rows from the group table summaries that meet the HAVING criteria will be included in the final result table.

Then any ordering clause apply, and finally the SELECT clause applies.

Page T-1: Terminology



Base table
Result table
Working table, intermediate table

Implicit group table
Explicit group table
Group table result row

Please drop me an email if you noticed any errors in this module. I'd also appreciate reading your comments, criticisms, and or suggestions as to how this module could be improved.

Thanks,

bil

That’s All