Divide

DreamHomes

Get a cross-join

create view ALLV3R AS

(select clientno, propertyno

from client, propertyforrent where rooms = 3);

ALLV3R $= \sigma_{rooms=3} \pi_{clientno,\ propertyno}$(clientXpropertyforrent)

Get the actual viewings

```
create view VIEW3ROOMS AS
(select clientno, propertyno
from client
join viewing using (clientno)
join propertyforrent using (propertyno)
where rooms = 3);
```

View3rooms=

$\Pi_{clientno,propertyno}$(clienttviewing⊠$\sigma_{rooms=3}$propertyforrent)

Please note, I'm using CAPITAL T in Wingdings 2 font as a join symbol.

- To get the dividend, we take everything from the cross-join (ALL3VR and join it, where there is a join, to everything in View3rooms.
- Any client that has no match in View3rooms hasn't viewed at least one of the properties:

```
Select distinct A.clientno
from allv3r A
left join view3rooms V
on (A.clientno = V.clientno and
    A.propertyno = V.propertyno)
where V.clientno is null;
```

To get the final list…

```
Select clientno from allv3r
  minus
(select distinct A.clientno from
  allv3r A left join view3rooms V
  on (A.clientno = V.clientno and
  A.propertyno = V.propertyno)
  where V.clientno is null);
```

Aggregate fields

And working with more than one row.

Non-base fields

- Derived fields
- Sum
- Average
- Count
- Summing group fields

BUILDER2 Schema being used

- It has a table called stock, with the following description:
  - STOCK(stock_code, stock_description, supplier_id, unit_price, unitCostPrice, stock_level, reorder_level)

Where

- the stock_code uniquely identifies each stock item
- The stock_description is the name of the stock item
- The supplier_id is a number identifying the supplier
- The unit price is what we charge for one item of this stock type.
- The UnitCostPrice is what we pay for one item of this stock type.
- The stock_level is the amount of this stock we have.
- The reorder_level is the minimum number that we want – we want to reorder if the stock falls below this level.

Summing within a row

- This does not count as an aggregate – just as a derived field:
  - To derive data within a row, just do the calculation as another field in the ‘select’:
    - Select stock_code, (Stock.qtyinstock * unitCostPrice) as StockSpend from stock;
  - This is the same as modifying output with a function;

To aggregate over > 1 row

- Count rows
  - How many booktitle rows do we have?
  - How many booktitle rows do we have that are in category 3?
- Sum over rows
  - How many physical items of stock do we have?

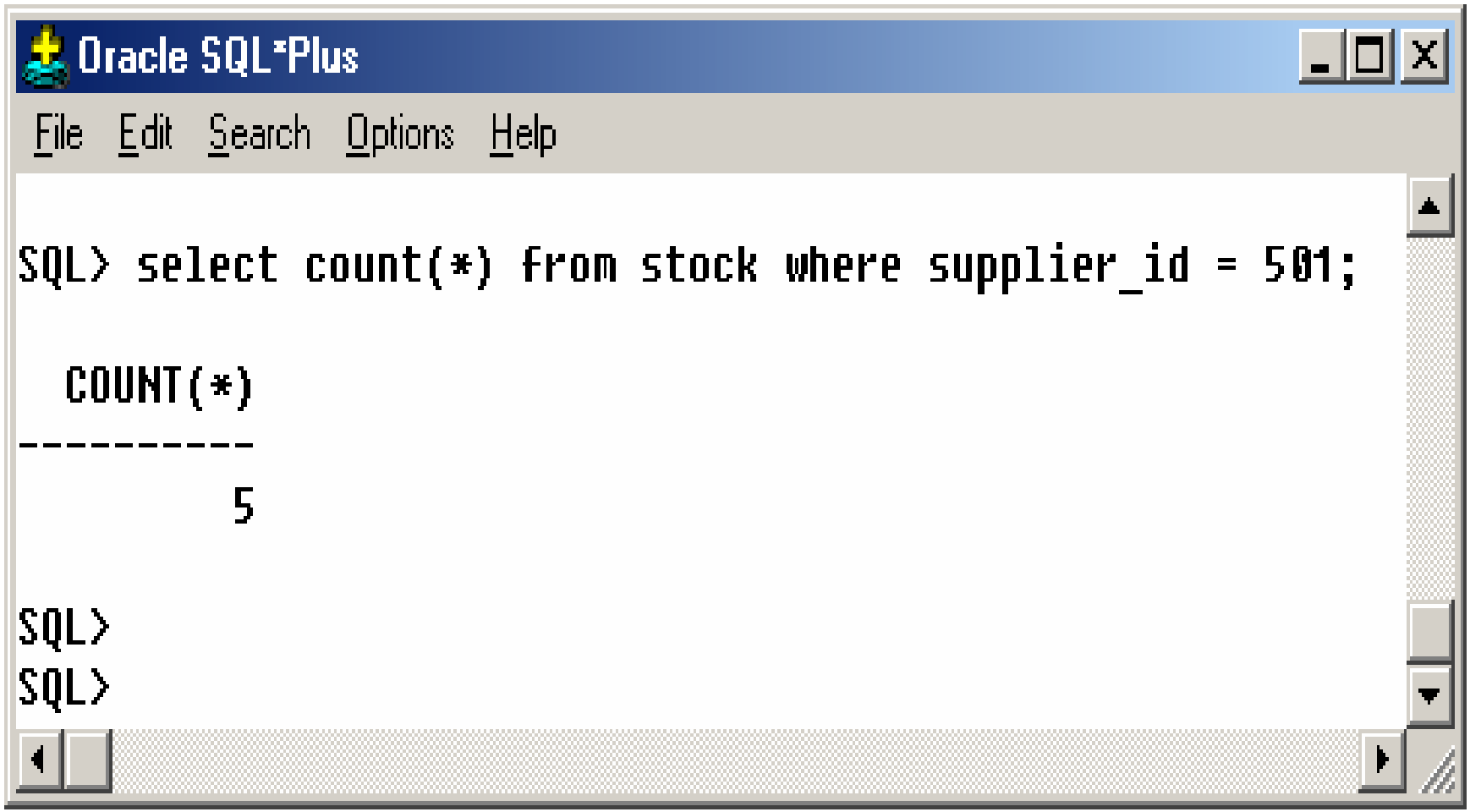
```
SELECT
  SUM(stock_level)
FROM stock;
```

$\mathcal{I}_{\text{stock\_level}}(R)$

Count the rows

- The COUNT function counts rows:
  - SELECT COUNT(*) FROM stock;
  - This counts the number of rows in stock.
- I could also count the number of items of stock that were supplied by a single supplier:
  - SELECT COUNT(*) FROM stock WHERE supplier_id = 501;
  - $\sigma_{supplier\_id=501}\mathcal{L}_{count}(STOCK)$

Outcome

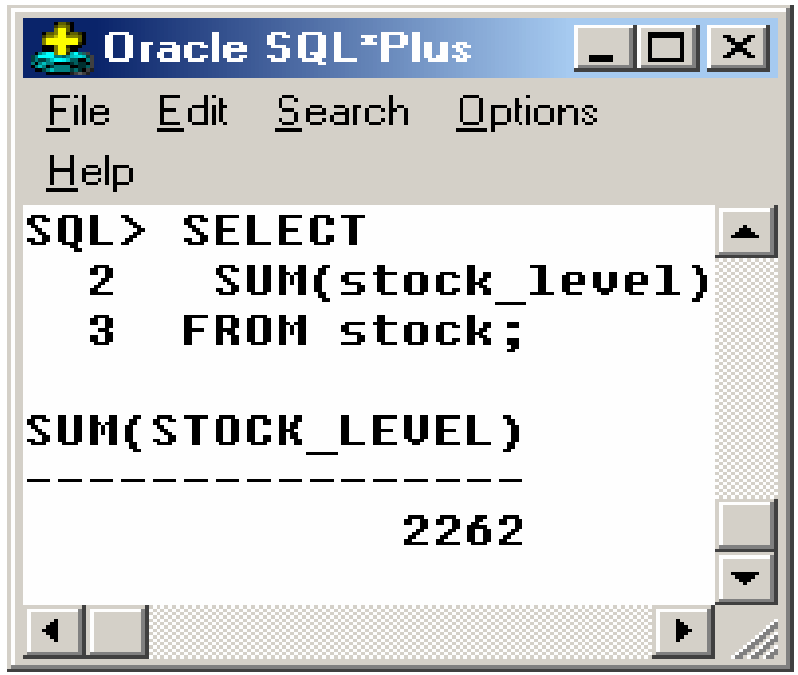

Oracle SQL*Plus
File Edit Search Options Help
SQL> select count(*) from stock where supplier_id = 501;
COUNT(*)
5
SQL>
SQL>

Likewise, I can sum things

- How many physical items of stock have I got?

```
SELECT
    SUM(stock_level)
FROM stock;
```

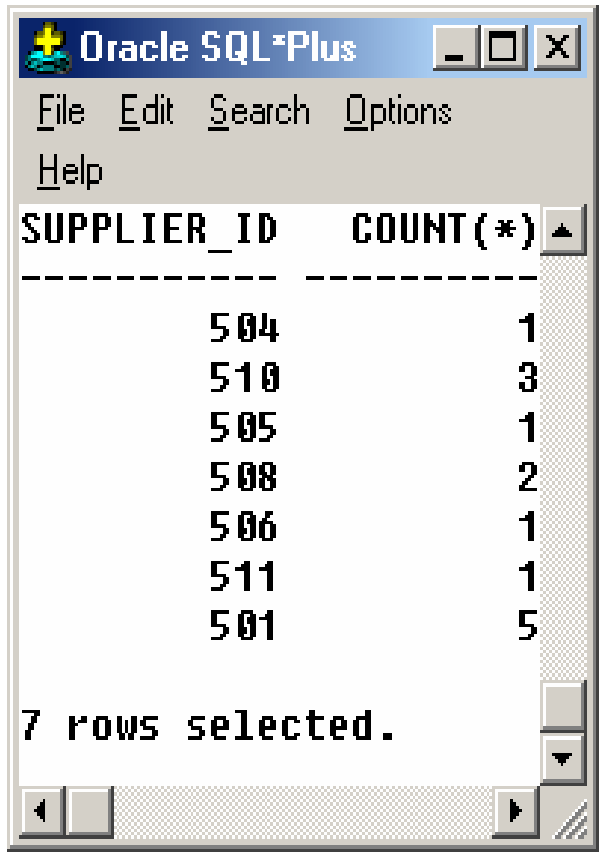
$\mathcal{L}_{sum(stocklevel)}(STOCK)$



Grouping

- Sometimes we need to have sub-groups.
- This allows us to get subtotals.
- This is done by using the ‘GROUP BY’ clause.
- We can count all stock types for each supplier:

```
SELECT supplier_id, COUNT(*)
 FROM stock
 GROUP BY supplier_id;
```



SELECT DISTINCT

- SELECT DISTINCT supplier_id FROM stock
  - This returns a list of supplier_ids that appear in the stock table, eliminating duplicates.

Examples

- **select distinct category from booktitle;**
- **select count(*) from booktitle;**
- **select max(category) from booktitle;**
- **select min(category) from booktitle;**
- **select avg(category) from booktitle;**
- **select count (*), category from booktitle group by category;**

Summary Queries

- Summary queries involve multiple rows in a table.
- The rows are GROUPED, using the GROUP BY clause.
- A function (MIN, MAX, AVG, SUM, COUNT) operates on one of the columns of the table.
  - E.g. select count(*) from stock;
  - This will return the number of stock ROWS in the table.
  - Select min(unitprice) from stock;

More detail…

- As well as grouping by one field, you can group by more than one:
- You can have sub-totals within sub-totals.
- This can be:
  - either nested (rollup),
  - or separately (cube)
  - *See later for examples.*

When to use CUBE

- Use CUBE if the rows are not hierarchical – i.e. the supplier id is completely independent of the category id, so we may need a total for each category id and for each supplier id.
  - I.E. if it is in a grid.

When to use ROLLUP

- When the groups you are sorting are nested:
  - If I want to see all of the orderlines in an order
  - If I want to see total number of students per stage and per program
    - As a stage is part of a program, we can nest the totals.

How do I?

- Make decisions based on group conditions?
- E.g To find
  - What is the average number of loans per booktitle?
  - Which authors have more than 2 books?

```
SELECT author, COUNT(*) FROM pobyrne.booktitle
GROUP BY author
HAVING COUNT(*) >2
```

Sub-queries

- Sometimes, you want to query a result set of another query:
- E.g.
  - How many stock items cost more than the average cost?
    - 1 query gets the average
    - The second query gets the products that cost more than the average

Query with sub-query

```
select stock_description, unitprice from
   stock
where unitprice >
(select avg(unitprice) from stock);
```

- In this case, the answer to the inside query is a single value or *scalar*.

Vector sub-queries

- How many customers do I have who have no orders with us?

```
select companyname from
  customers
where customerId not in
(select customerid from orders);
```

The inner query returns a set of rows, or a *vector.*

Where and Having

- The ‘WHERE’ allows filtering on rows.
  - e.g. *Show all stock rows where the price of the stock item is > €5,000*
- The ‘HAVING’ allows filtering on summary rows.
  - e.g. *Show all stock items supplied by a supplier who provides more than €10,000 worth of stock.*

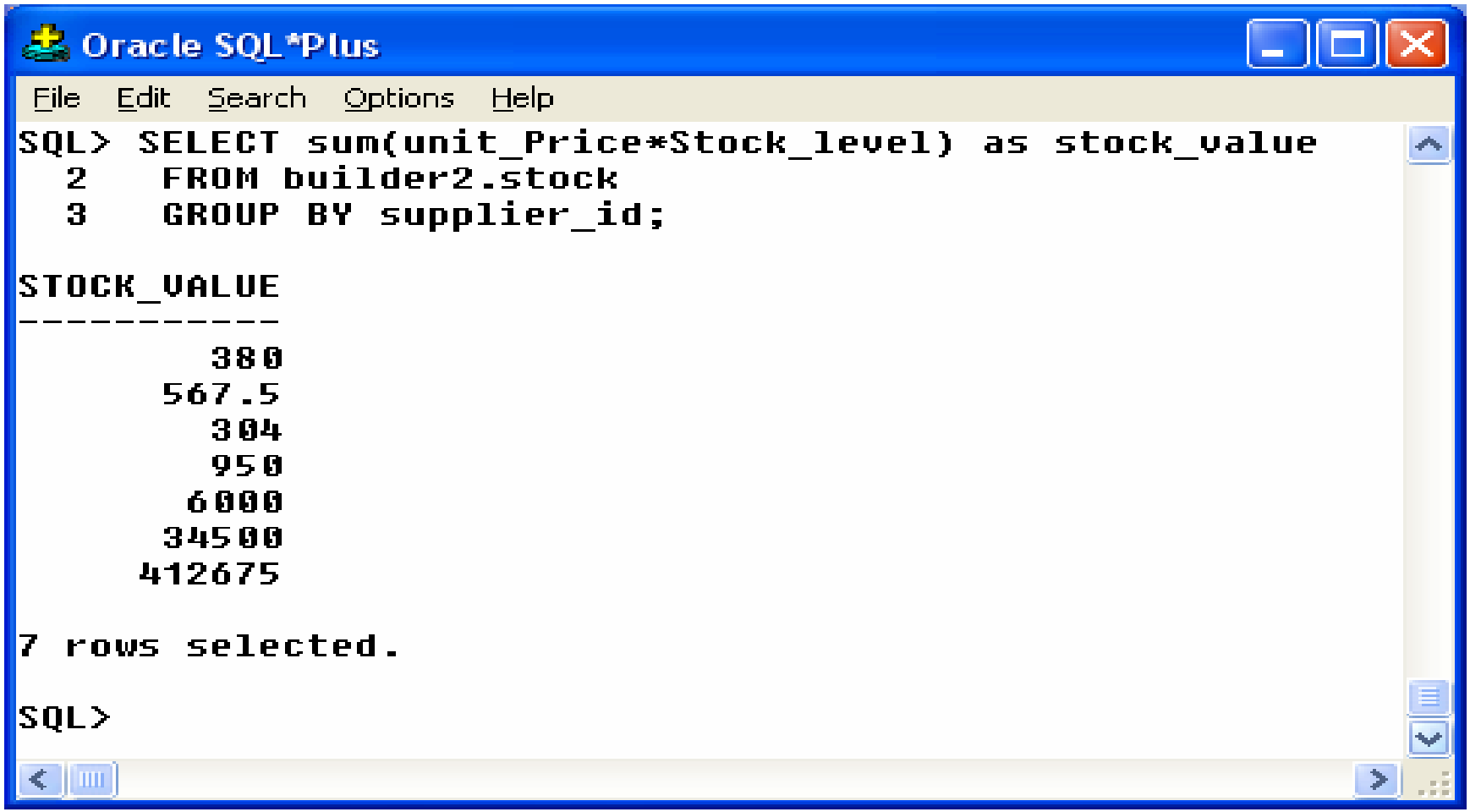
Nesting queries

- Aggregates can only work on a single pass through a set of rows.
- Which supplier's stock is the most valuable to us?
  - A stock item's value is the unitPrice*Stock_level
  - Find out how much each supplier's stock is worth.
  - Find out which is the supplier whose stock is the most valuable.

Inner query – Find the value of stock for suppliers

```
SELECT
 sum(unit_Price*Stock_level) as
 stock_value
FROM builder2.stock
 GROUP BY supplier_id;
```

Result

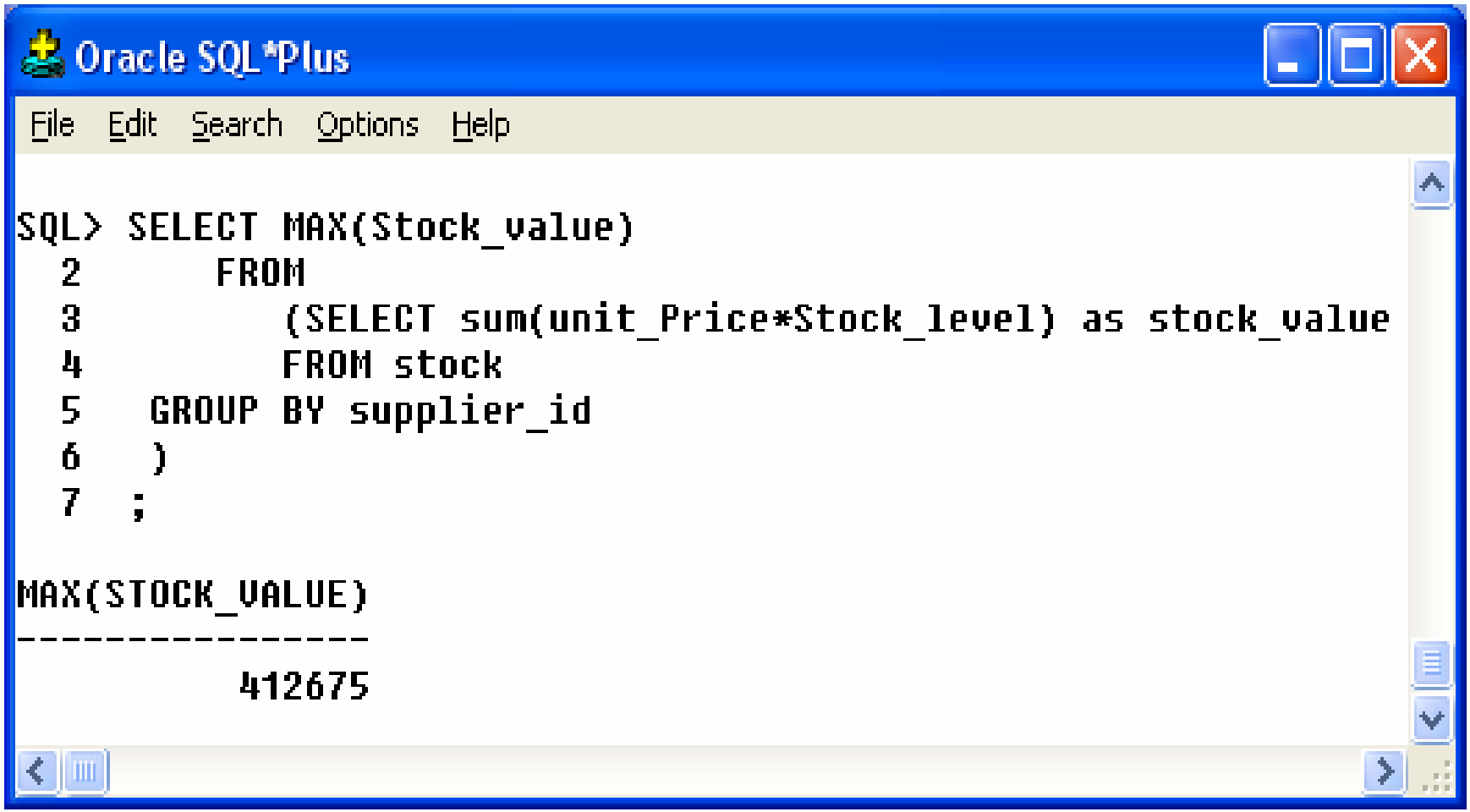

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT sum(unit_Price*Stock_level) as stock_value
2 FROM builder2.stock
3 GROUP BY supplier_id;
STOCK_VALUE
380
567.5
304
950
6000
34500
412675
7 rows selected.
SQL>

Next…

- Get the maximum value from the above:

```
SELECT MAX(Stock_value)
  FROM
      (SELECT
sum(unit_Price*Stock_level) as
stock_value
      FROM stock
GROUP BY supplier_id
);
```

Result

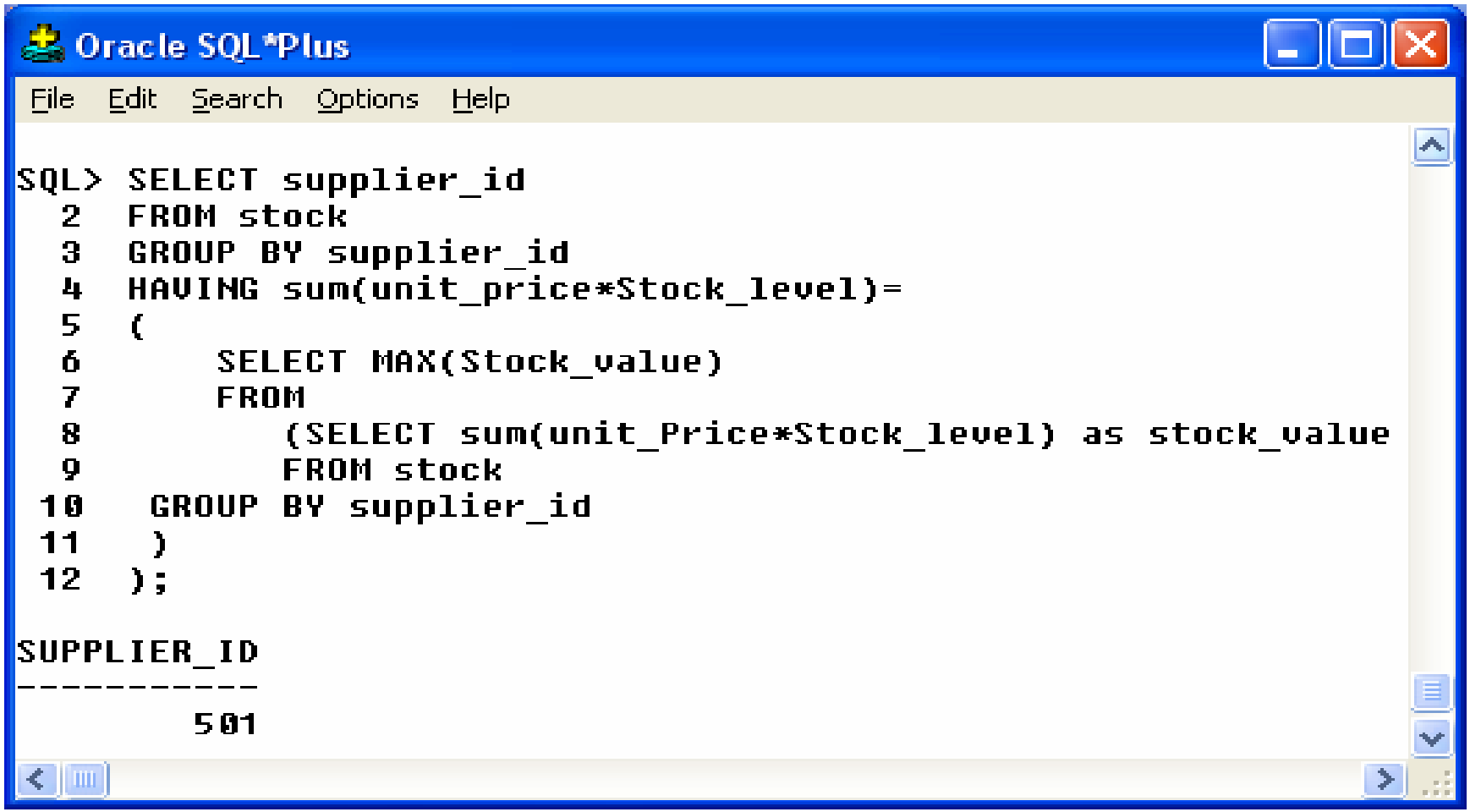

Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT MAX(Stock_value)
2 FROM
3 (SELECT sum(unit_Price*Stock_level) as stock_value
4 FROM stock
5 GROUP BY supplier_id
6)
7 ;
MAX(STOCK_VALUE)
412675

Next…

- Get the supplier whose stock value has the same value as the maximum stock value:

```
SELECT supplier_id
FROM stock
GROUP BY supplier_id
HAVING sum(unit_price*Stock_level)=
(
    SELECT MAX(Stock_value)
    FROM
        (SELECT sum(unit_Price*Stock_level) as
  stock_value
        FROM stock
  GROUP BY supplier_id
  )
);
```

Result


Oracle SQL*Plus
File Edit Search Options Help
SQL> SELECT supplier_id
2 FROM stock
3 GROUP BY supplier_id
4 HAVING sum(unit_price*Stock_level)=
5 (
6 SELECT MAX(Stock_value)
7 FROM
8 (SELECT sum(unit_Price*Stock_level) as stock_value
9 FROM stock
10 GROUP BY supplier_id
11)
12);
SUPPLIER_ID
501